INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

WILEY

INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

Intl. Trans. in Op. Res. 29 (2022) 3435–3463
DOI: 10.1111/itor.13120

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

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

[a] *Department of Statistical Sciences, Sapienza University of Rome, P.le Aldo Moro 5, Rome 00185, Italy*
[b] *Department of Statistical Sciences and Operational Research, University of Seville, Seville, Spain*
E-mail: lavinia.amorosi@uniroma1.it [Amorosi]; puerto@us.es [Puerto]

Received 10 May 2021; received in revised form 30 September 2021; accepted 12 January 2022

Abstract

This paper presents a new two-phase algorithm for the bi-objective minimum spanning tree (BMST) problem. In the first phase, it computes the extreme supported efficient solutions resorting to both mathematical programming and algorithmic approaches, while the second phase is devoted to obtaining the remaining efficient solutions (non-extreme supported and non-supported). This latter phase is based on a new recursive procedure capable of generating all the spanning trees of a connected graph through edge interchanges based on increasing evaluation of non-zero reduced costs of associated weighted linear programs. Such a procedure exploits a common property of a wider class of problems to which the minimum spanning tree (MST) problem belongs, that is the spanning tree structure of its basic feasible solutions. Computational experiments are conducted on different families of graphs and with different types of cost. These results show that this new two-phase algorithm is correct, very easy to implement and it allows one to extract conclusions on the difficulty of finding the entire set of Pareto solutions of the BMST problem depending on the graph topology and the possible correlation of the edge costs.

Keywords: multiple objective programming; combinatorial optimisation; minimum spanning tree; two-phase methods

1. Introduction

In many real-world problems, different goals, often in conflict with each other, have to be taken into account simultaneously. Several examples of this appear in different application fields. In the transportation sector, one finds conflicting interests between efficiency, cost-effectiveness and security. In the telecommunications sector, energy saving and reliability should be combined. Finally, to mention some others, in the construction industry cost and environmental impact, or in finance expected return and risk have to be considered simultaneously. The increasing necessity of dealing

*Corresponding author.

with this type of problem has influenced the strong development of the field of multi-objective optimisation in recent years. The reader is referred to Salamirad et al. (2021), Amaruchkul (2021), Xiang et al. (2021), Erdoğdu and Karabulut (2021), Bissoli et al. (2021) and Queiroz and Mundim (2020) for very recent applications. In this work, we focus on a particularly appealing problem: the bi-objective minimum spanning tree (BMST) problem that has applications in different contexts. For example, in the energy industry, for planning efficient distribution systems or in the telecommunications sector, for designing networks whose topological structure ensures connectivity between all nodes. In the context of the energy efficiency of the networks, both quality of the service and energy consumption have to be taken into account. This leads to a bi-objective problem that, in its basic form, can be formulated as a BMST problem.

The main contributions of this paper can be summarised as follows:

- it provides a new approach to solving the BMST problem, based on a new enumerative recursive procedure (see Amorosi, 2018) applicable to a wider class of combinatorial optimisation problems defined on graphs, able to generate all feasible solutions and that can be exploited in the second phase for obtaining a complete set of efficient solutions;
- it gives a comparison between three alternative methods, and open source solvers, adopted to implement the first phase: the dual variant of Benson's algorithm (Ehrgott et al., 2012), by means of Bensolve (Weißing and Löhne, 2014), the lifted weight space approach, by means of PolySCIP (Schenker and Strunk, 2016), and the Prim's algorithm embedded in a weighted sum approach (Przybylski et al., 2010a);
- it proposes a new enumerative recursive procedure based on an analysis of reduced costs, first introduced in Amorosi (2018) and in Amorosi and Ehrgott (2018) for the bi-objective integer minimum cost flow problem, able to generate all the spanning trees of a connected graph;
- it contains extensive computational results obtained testing the algorithm on different problem instances, including complete and grid graphs and comparisons with the most recent two-phase method for this class of problem by Steiner and Radzik (2008) and with the one by Di Puglia Pugliese et al. (2018) that is the most recent one among the methods designed for the multi-objective version of the MST problem;
- it reports a complete computational experience to analyse the complexity of solving the BMST problem, depending on the type of edge costs considered, and how the proportion of non-supported efficient solutions depends on the graph topology and type of costs (an electronic summary includes detailed information of all tests).

This paper is organised as follows: in Section 2 some preliminary concepts are presented, Section 3 discusses the literature focusing on two-phase methods. In Section 4, the three alternative approaches for the first phase are described. A new recursive procedure able to enumerate all spanning trees of a connected graph, and its adaptation in the second phase is presented in Section 5. Finally, Section 6 reports experimental results.

2. Bi-objective minimum spanning tree problem

Let $G = (N, E)$ be an undirected graph with node set N, with $|N| = n$, and edge set E. Let $C = (c_1, c_2)$ be the coefficient matrix where c_1 and c_2 are two different integer cost vectors on the edge

set. Eventually, let $E(S)$ be the set of edges connecting nodes belonging to S, $\forall S \subseteq N: \ S \neq \emptyset$. The BMST problem is defined as

$$\min(y_1, y_2) = \min Cx = \min(\textstyle\sum_{e\in E} c_e^1 x_e, \sum_{e\in E} c_e^2 x_e) \tag{1}$$

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

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

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

In the multi-objective context, and thus in the bi-objective case, the feasible set in the decision space (or decision set) $X = \{x \in R^n : Ax = b, x \geq 0, \ x \in \{0, 1\}^n \quad \text{or } x \in Z^n\}$ 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.

Definition 1 (Efficient or Pareto optimal solution)**.** *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 (or point in the bi-objective case) $y^* = Cx^*$ is called non-dominated.*

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

Definition 3 (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 4 (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 remainder 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 $X_{NE} = X_E \backslash X_{SE}$. Their images in the objective space are indicated respectively by Y_{SN} and Y_{NN}. The computation of the set Y_{NN} is important because it usually represents the majority of the Pareto frontier and it contributes to the difficulty of the problem (see Ehrgott and Gandibleux, 2000). Indeed, from a geometrical point of view, the vectors corresponding to non-supported efficient solutions, are located in the interior of the convex hull of the feasible region in the objective space of the problem. Standard scalarisation techniques, like the weighted sum method, are not able to generate them and it is necessary to resort to more sophisticated methods. A detailed review of the ones proposed in the literature for the BMST is reported in Section 3.

Definition 5. *A set UB is an upper bound set of Y if $\forall y \in Y, \exists u \in UB : \forall i, \ u_i \geq y_i$*

Definition 6 (T-exchange/cyclic-interchange)**.** *Let T be a spanning tree of the graph G. A T-exchange is a pair of edges e, f where $e \in T$, $f \notin T$, and $T - e \cup f$ is a spanning tree. The weight of the T-exchange e, f is $w(f) - w(e)$, where w is the vector of the graph weights associated with the edges.*

Definition 7 (Fundamental cycle)**.** *Let T be a spanning tree of the graph G. The addition of any non-tree edge to the spanning tree T creates exactly one cycle. Any such cycle is a fundamental cycle of G with respect to the tree T.*

3. State of the art

Among the exact methods for solving the BMST problem proposed in previous works, it is possible to distinguish between generalisations of algorithms for the single-objective case and generic approaches for multi-objective combinatorial optimisation problems applied to this class.

3.1. Generalisations of single-objective algorithms

In Corley (1985), a generalisation of Prim's algorithm is presented. The idea of this generalisation 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. Hamacher and Ruhe (1994) present an enhancement of Corley's algorithm, excluding in each iteration subtrees which are non-efficient. The resulting algorithm is also discussed in Ehrgott (2005) and used in Zhou and Gen (1999) to evaluate a genetic algorithm proposed by the authors able to approximate the Pareto frontier. Knowles and Corne (2002) showed that using the algorithm presented in Hamacher and Ruhe (1994), some efficient spanning trees may not be generated and consequently some non-efficient spanning trees may also be produced. Serafini (1987) proposed a generalisation of Kruskal's algorithm. Also in Perny and Spanjaard (2005) generalisations of Prim's and Kruskal's algorithms are considered to determine the set of efficient spanning trees in accordance with a preference (binary) relation defined on the set of the graph edges. In addition, in Fernández et al. (2017) multi-objective spanning tree problems are analysed from the perspective of ordered weighted averaging operators.

3.2. Generic approaches for the BMST problem

Turning to the generic approaches for multi-objective combinatorial optimisation applied to the bi-criteria minimum spanning tree problem, both two-phase methods and multi-objective branch-and-bound methods have been studied. Ramos et al. (1998) presented a two-phase approach for the BMST problem that works in the objective space, proposing a branch-and-bound algorithm in the second phase. This algorithm evaluates search nodes according to ideal point associated with the subproblem in the corresponding node, in order to explore the search area defined by the non-dominated extreme points generated in the first phase (extreme points of the convex hull of feasible

solution vectors in objective space). The computational results reported in that paper show that this approach may not be practical for large instances.

Steiner and Radzik (2008) introduced another two-phase approach based on a ranking algorithm, the k-best algorithm by Gabow (1977). This k-best algorithm is applied in the second phase to explore the triangles identified by consecutive pairs of non-dominated extreme points. This paper also provides a comparison with the procedure by Ramos et al. (1998) that shows the better performance of their approach in terms of computational time and instance sizes that are solvable. Moreover, a heuristic enhancement of the k-best algorithm is also proposed which, in general, speeds up the running time.

Sourd and Spanjaard (2008) proposed an improved branch-and-bound algorithm applied to the BMST 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-and-bound tree, while the improving ones are the solutions that are not dominated by the upper bound set. Experimental results show that this algorithm is able to solve instances with up to 400–500 nodes, also improving the algorithm by Steiner and Radzik (2008) in terms of instance sizes that are solvable.

Leitner et al. (2015) applied the ϵ-constrained method to the bi-objective prize-collecting Steiner tree problem, comparing it with other scalarisation techniques and two-phase methods. This was the first study on exact approaches to solve that bi-objective problem proposing some new generic acceleration methods to recycle the information gleaned during these iterative solution procedures. The reader is also referred to LabbÃ© et al. (2021) for another recent reference to multilevel decision problems on trees.

Di Puglia Pugliese et al. (2015) provided the extension of a dynamic programming formulation for the spanning tree problem to solve the multi-objective minimum spanning tree problem. The underlying idea is to model the spanning tree structure by means of states and transition between states, defining a state-space. Their experimental results show that the proposed algorithm is able to solve instances up to 100 nodes and 5 objective functions.

In a recent paper, Di Puglia Pugliese et al. (2018) present a further algorithm for the multi-objective minimum spanning tree problem. This is based on a labelling algorithm for the multi-objective shortest path problem in a transformed network. Additional restrictions to the minimal paths are inserted in order to obtain a one-to-one correspondence between trees in the original network and minimal paths in the transformed one. Computational results related to instances up to 14 nodes and 4 criteria show that the proposed algorithm outperforms the one in Di Puglia Pugliese et al. (2015) in solution time on this type of instances. Focusing on the bi-objective case, although only on small size graphs, the proposed algorithm performs better, with the exception of the instances of 14 nodes, than the one by Steiner and Radzik (2008), which in turn performs better than the algorithm introduced in Di Puglia Pugliese et al. (2015).

3.3. Two-phase methods

The two-phase method was introduced by Ulungu and Teghem (1994b). This is a general framework for the exact solution of multi-objective combinatorial optimisation (MOCO) problems. It

consists in generating a complete set of efficient solutions in two steps: in the first one, supported efficient solutions are generated, whereas in the second one, using information from the first phase to limit the exploration area in the outcome set, a complete set of non-supported efficient solutions is found.

One can find, in the literature, several two-phase algorithms proposed for different MOCO problems: the bi-objective knapsack problem (Ulungu and Teghem, 1994a; Visée et al., 1998); the assignment problem (Przybylski et al., 2008), also including three objectives (Przybylski et al., 2010b); the bi-objective set covering problem (Prins et al., 2006); the bi-objective integer network flow problem (Raith and Ehrgott, 2009b) and the bi-objective minimum spanning tree problem (Steiner and Radzik, 2008), to mention a few.

Several techniques can be used for the generation of supported Pareto optimal solutions (for example, the weighted sum method or the lexicographic method, see Ehrgott and Gandibleux (2000) and Ehrgott (2006)). On the contrary, an efficient general procedure for generating non-supported Pareto optimal solutions does not exist, due to the lack of a theoretical characterisation of those solutions at the current state of the art and, for the BMST problem, the non-connectedness of efficient solutions (as stated in Ehrgott and Klamroth (1997) and Gorski et al. (2011), which study connectedness of efficient solutions of MOCO problems).

In the bi-objective case, the prevalent strategy used in the second phase to generate them is the restriction of the exploration area in the outcome set. This strategy reduces the search to the triangles that can be built considering consecutive pairs of non-dominated extreme points sorted in increasing order according to the first objective (see Steiner and Radzik, 2008 and Raith and Ehrgott, 2009b). In each of these triangles, the exploration to find non-dominated points depends on the specific problem under consideration.

Restricting oneself to the two-phase algorithms for the BMST problem, in the most recent one proposed by Steiner and Radzik (2008), the first phase consists in Kruskal's algorithm embedded in a dichotomic search and, as regards the second phase, a k-best algorithm by Gabow (1977) is applied to produce in an ordered manner feasible spanning trees of a graph $G = (V, E)$. Starting from the initial optimal spanning tree T (the one with minimum cost), a T-exchange of two edges e, f such that $e \in T$, $f \in E \setminus T$ of minimum weight, is applied. In this way, the second best spanning tree is generated. At a generic iteration j, for each of the spanning trees generated until the iteration $j-1$ ($T_1, \ldots, T_{j-1}$), two lists of edges are created. The first one (IN) is the list containing the edges in T that must remain in T and the second one (OUT) is the list of edges that are not in T and that must remain out of T. At each iteration, a T-exchange e, f, is applied such that $e \in T \setminus IN$, $f \in E \setminus T \setminus OUT$ and e, f have the smallest possible weight, among all the possible T-exchanges applicable to the previous spanning trees generated until the current iteration. The newly generated spanning tree is the one with the minimum possible deterioration of the objective value with respect to the previous one.

In this paper, we present a new two-phase method for the BMST problem, which exploits a novel recursive procedure, we proposed for enumerating feasible solutions of the problem in the second phase in order to generate efficient solutions. Unlike the algorithm by Gabow (1977), adopted in the second phase by Steiner and Radzik (2008), this procedure is not an "ad hoc" procedure for the MST but it is based on the analysis of non-zero reduced costs, exploiting a characteristic of the basic feasible solutions of the problem, that is of being spanning trees. This property, as it is well known, is common to a wider class of problems (single- and multi-commodity network flow problems). Thus,

even if in this paper we focus on the BMST, the working principle of this procedure can be easily extended to other combinatorial optimization problems. Moreover, as regards the implementation of the first phase, we present an experimental comparison between different strategies and available tools for multi-objective programming. In particular, we adopted both mathematical programming and algorithmic approaches by using two open-source solvers, Bensolve and PolySCIP, and by coding Prim's algorithm embedded in a weighted sum approach.

4. Finding supported efficient solutions

In this section, we describe three alternative strategies to implement the first phase of the two-phase algorithm proposed for the BMST problem. The first one is the weighted sum method (see for example Ehrgott, 2006), which is a scalarisation 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 bi-objective integer linear programming problem.

We adopted this technique by means of two different approaches: a mathematical programming one, resorting to the flow formulation of the BMST problem and a greedy one using Prim's algorithm embedded in a dichotomic search in the objective space. The third strategy, adopted for the first phase, is the dual variant of Benson's algorithm proposed by Ehrgott et al. (2012), using the geometric duality theory for multi-objective continuous linear programming developed by Heyde and Löhne (2008). To apply it to the MST problem, it is necessary to consider a different formulation of the problem that ensures its continuous optimal solutions are integers. At this point, we resort to a formulation by Kipp-Martin (1991) characterised by the integrality property.

4.1. The weighted sum method

The weighted sum method can be applied to any valid formulation of the BMST problem. Here, it has been applied to its flow formulation (see Magnanti and Wolsey, 1995) reported below.

Flow formulation for the BMST problem:

$$\min \left(\sum_{e \in E} c_e^1 x_e, \sum_{e \in E} c_e^2 x_e \right) \tag{5}$$

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

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

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

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

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

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

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

This formulation is characterised by a polynomial number of constraints. Through this formulation, the BMST problem is represented as a single-commodity flow problem in which node 1 is considered as a source of $n-1$ units of flow (6) and the rest of the network nodes are sinks each requiring 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 considered in the solution. The flows 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 to formulations (5)–(12) to generate 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. Analogously, the same procedure can be applied using Prim's algorithm, instead of an LP solver, to iteratively solve the weighted sum problem and thus to generate the supported efficient solutions (see, e.g., Przybylski et al., 2010a).

4.2. Benson's algorithm on the Kipp-Martin formulation

An alternative approach to implement the first phase is to be found in use of the dual variant of Benson's algorithm (Ehrgott et al., 2012). This is a solution method for multi-objective linear programming problems capable of generating all the extreme efficient solutions, that is, the vertices of the feasible region. The main idea of the primal version by Benson (1998) is to consider another polyhedron $\bar{Y}$, opportunely defined so that it has the same set of non-dominated points as Y, but where it is easier to identify the set of all non-dominated 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 Kipp-Martin (1991), a linear programming formulation for the MST problem, based on a totally dual integral system, was proposed and it was 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 thus obtained will be an integer. In this formulation, reported below, new non-negative variables $z_{kij} \forall k \in V, \ (i,j) \in E$, associated with triplets of nodes, are introduced together with additional constraints that guarantee the covering of all nodes without the presence of cycles. More precisely, this formulation models an arborescence for each node $k \in V$. For $k \in V$ and $(i,j) \in E$, the variables z_{kij} and z_{kji} respectively indicate whether or not arc (i,j) and (j,i) belong to the arborescence rooted at k.

Kipp-Martin formulation of the BMST problem:

$$\min \left(\sum_{e \in E} c_e^1 x_e, \sum_{e \in E} c_e^2 x_e \right) \tag{13}$$

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

$$z_{kij} + z_{kji} = x_e, \qquad k = 1, \ldots, n, \;\; e = (i, j) \in E \tag{15}$$

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

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

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

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

Constraints (15), (16) and (17) guarantee that the solution does not contain undirected cycles. Indeed, if there is a cycle of undirected edges containing vertex k, then by constraints (17) there is a corresponding cycle of directed edges defined by z_{kij}, which also contains vertex k. However, there cannot be a cycle of directed arcs which contains vertex k. This is impossible by constraints (15) and the cycle cannot be directed. If the cycle is not directed, then there is at least one vertex i with two cycles of directed edges containing k, which are directed out of i. This is impossible by constraint (16). Thus, there are no cycles in the solution and the solution is spanning tree by (14).

5. A recursive algorithm to complete the set of efficient solutions

In the second phase of the algorithm, to generate 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 extreme points and the corresponding local nadir point, as shown in Fig. 1, where the horizontal axis reports the values of the first objective function (y_1) and the ordinates refer to values of the second objective function (y_2).

At each iteration of the algorithm, one of these triangles is analysed, 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 (at implementation level any formulation with the objective function defined as in (20) is suitable):

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

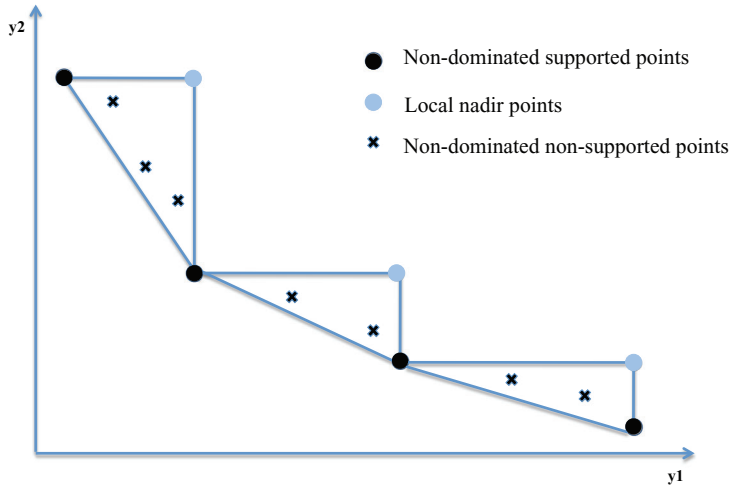


Fig. 1. Example of triangles in the objective space.

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

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

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

where λ_1 and λ_2 are the weights associated with each objective function. These weights are defined as in Raith and Ehrgott (2009b): 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}.

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 an analysis of the reduced costs associated with an initial optimal spanning tree, that is able to generate all the other spanning trees of a given network in a given order. This algorithm, unlike the k-best algorithm by Gabow (1977), does not produce spanning trees in strict ranking order with respect to the scalarised 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 terms of objective value. This algorithm is very easy to implement and it can be applied to each triangle starting indifferently from any of the optimal vertices, x^{i+1} or x^i, of the weighted sum problem (20)–(23).

5.1. Procedure to generate all the spanning trees of a graph

The procedure proposed to generate all the spanning trees of a graph works considering the reduced costs associated with the edge variables of the initial optimal solution. The reduced cost of each edge is computed as the difference between its cost and the highest cost along the (unique) path between its endpoints in the spanning tree. The reduced costs are then sorted from the lowest one to the highest one and analysed in this order.

More in detail, 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 to their integer costs, from the highest to the lowest. For each of these edges, a new spanning tree is generated by swapping (exchanging) edge e with the edge under consideration. The order in which the edges in the fundamental cycle are exchanged with the edge 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 spanning tree obtained is exactly the second best with respect to the current weighted objective function. Indeed, this new spanning tree is generated by inserting in the initial one, the edge with minimum non-zero reduced cost and removing from the fundamental cycle thus created, the edge with maximum cost. Therefore, the corresponding deterioration of the objective function is minimal. When all the edges in the fundamental cycle created by edge e are exchanged with e, the second non-zero reduced cost is analysed, 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 edge $e^{'}$). In this case, after each exchange, starting from the new spanning tree $T^{'}$ obtained, the first non-zero reduced cost, associated with edge e, has to be considered again, implementing all the possible exchanges of 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 lowest to the highest, and all their possible combinations. Therefore, the resulting algorithm consists in a recursive procedure that is summarised in the pseudocode of Algorithm 1.

In order to prove the correctness of the recursive procedure **find_feasible_trees**, we recall Theorem 1 from Gross and Yellen (2006).

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 tree $T^{'}$ starting from 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 a unique optimal solution T^*. Procedure **find_feasible_trees** examines all the feasible combinations of T^*-exchanges.*

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

INPUT: graph $G = (N, E)$, integer costs vector c, optimal solution T^*, vector of the non-zero reduced costs $\bar{rc}^* = \{rc_1^*, rc_2^* \dots rc_K^*\}$ (sorted from the lowest to the highest) where K is the number of non-zero reduced costs, and the list $FT = \{T^*\}$ containing at the beginning only the initial optimal spanning tree T^*.

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

$it = 1$
while $it \leq K$ **do**
  Let e be the edge not in T^* corresponding to the element $rc_{it}^* \in \bar{rc}$ ($\bar{rc}_1^*$ 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 highest to the lowest) the edges of the cycle C^* different from e
  for $e^* \in C^*$ with $e^* \neq e$ in decreasing order of cost **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.

Proof. The proof relies on showing that all feasible T^*-exchanges obtained from edges with non-zero reduced costs can be generated. This is ensured because the algorithm performs recursive calls from each found spanning tree and the remaining not considered edges with non-zero reduced costs. In this way, all possible configurations of feasible trees with non-zero reduced costs are generated in the recursive calls. In the first call, **find_feasible_trees** first examines all the fundamental cycles associated with an edge out of T^* with non-zero reduced cost. Next, the procedure combines the previous T^*-exchange with a second edge with non-zero reduced cost and so on, generating all possible feasible trees. With this mechanism, all the feasible combinations of T^*-exchanges generated by the first two fundamental cycles are examined. At a generic stage k of the algorithm, the non-zero reduced cost rc_k^* is considered and with the same mechanism all feasible combinations of T^*-exchanges generated by the fundamental cycles associated with the first $k - 1$ non-zero reduced costs with the T^*-exchanges associated with the kth 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. In this way, all possible combinations of edges with non-zero reduced costs are considered and thus, all feasible 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 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^*. 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. □

Remark 1. It is worth observing that using the recursive procedure above, it is possible to generate the same spanning tree more than once. We also observe that if the optimal solution of the minimum spanning tree problem is not unique, we can proceed including in the analysis also zero reduced costs associated with variables out of the basis and the result still holds.

5.2. The recursive procedure adapted to the second phase

The recursive algorithm described in Section 5.1 is used to generate the remaining efficient solutions in the second phase. In each triangle, starting from one of the two extreme efficient solutions corresponding to the non-dominated extreme points 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 a non-dominated point inside the considered triangle, of the weighted sum problem (20)–(23). For the first triangle, this upper bound is defined (see Raith and Ehrgott, 2009b) 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 detail, when a new spanning tree is produced, the corresponding point in the objective space is computed. If this point is non-dominated by points found so far 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 Raith and Ehrgott, 2009a):

$$\Delta = \max\{\lambda_1(y_1(x^{i,j+1}) - 1) + \lambda_2(y_2(x^{i,j}) - 1), \quad j = 0, \ldots, r\} \tag{26}$$

where r is the number of efficient solutions found in the triangle under consideration and it is assumed that $x^{i,0}, x^{i,1}, \ldots x^{i,r+1}$ are sorted in increasing order with respect to the first objective ($x^{i,0} = x^i$ and $x^{i,r+1} = x^{i+1}$ are the two extreme efficient solutions associated with the triangle). Figure 2 shows an example of this update. It represents a given triangle defined by two consecutive extreme non-dominated points (the dark blue ones) and the corresponding local Nadir point (the black cross). The initial upper bound, to start the recursive procedure in this triangle, is represented by the constant term associated with the orange dotted line parallel to the line joining the two extreme non-dominated points and passing through the point defined by Equation (25), that is the point obtained subtracting one unit to each component of the local Nadir point. The light blue crosses represent the new non-dominated points generated up to the current stage of the recursive procedure applied to the given triangle. Let us consider the set composed of these two new non-dominated points and the two initial extreme non-dominated points, sorted in increasing order with respect to the first objective (y_1). For each pair of consecutive points, sub-triangles as the ones shown in Fig. 2 with light blue dotted lines, are built. For each of them, the corresponding

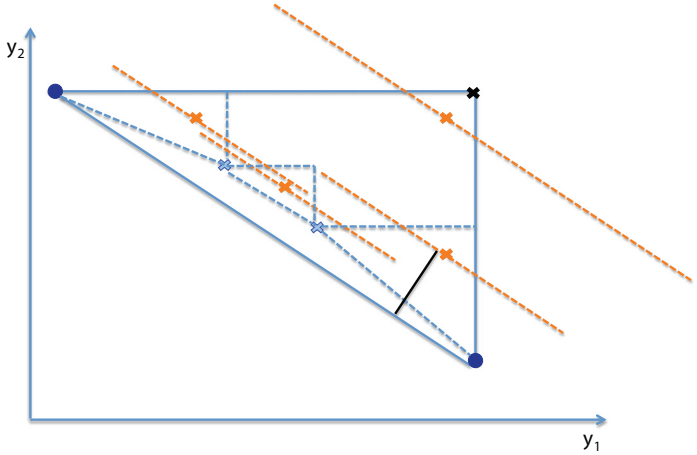


Fig. 2. Example of bound update during the recursive procedure.

local Nadir point is computed and, from it, the point obtained subtracting one unit to each of its components (orange crosses inside the sub-triangles in Fig. 2). Similar to the initial upper bound, the lines passing for each of these points so determined and parallel to the line joining the two extreme non-dominated points are considered (orange dotted lines in Fig. 2). The new upper bound defined by Equation (26) corresponds to the constant term of the line, among these latter, that has maximum distance from the line joining the two extreme non-dominated points. In the example of Fig. 2, this is the line whose distance from the hypotenuse of the triangle is represented by the black segment.

We should observe that formula (26) can be adopted even if the feasible solutions found by means of the recursive procedure are not generated in strict ranking order. Indeed, representing this procedure by means of a tree whose root is the initial optimal basic solution, as the analysis of the reduced costs is made in order from the lowest to the highest one in absolute value, at each branch of the tree certainly the value of the objective function will worsen. Thus, it is possible to adopt formula (26) to limit the generation of feasible solutions at each branch without missing any potentially efficient solution. The algorithm terminates when all the triangles have been explored by means of this recursive procedure. The pseudocode of the second phase is shown in Algorithm 2.

In order to limit the recursion and not to generate all feasible spanning trees, as explained above, the recursive procedure **find_feasible_trees**, called by the algorithm **Two_phase_btree**, is slightly modified by introducing the additional parameter Δ as reported in Algorithm 3.

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

Proof. Let $1 \leq i \leq s-1$. 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 $x \in \mathcal{E}_i$.

Algorithm 2. Second phase

INPUT: Network $G = (N, E)$, integer costs vector c, list of extreme efficient solutions $[x^1, \ldots, x^s]$.
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
    $\Delta = \max\{\lambda_1(y_1(x^{i,j+1}) - 1) + \lambda_2(y_2(x^{i,j}) - 1), j = 0, \ldots, r\}, r + 1 =\mid \mathcal{E}_i \mid$
  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**$[G = (N, E), c, T^*(x^{i+1}), \bar{rc}^*(x^{i+1}), K, \mathcal{E}, \Delta]$ /*with $T^*(x^{i+1})$ and $\bar{rc}^*(x^{i+1})$ respectively spanning tree and vector of the non-zero reduced costs (sorted from the lowest to the highest) associated with x^{i+1} and Δ additional input parameter representing the upper bound to limit the recursion*/
  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 $\mathcal{E}_i$ $i = 1, \ldots s - 1$ **do**
  Remove potential non-efficient solutions
end for
OUTPUT: Complete set $\mathcal{E}^* = \cup_{i=1,\ldots,s-1}\mathcal{E}_i$ of efficient solutions.

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 triangle T_i that 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 solution 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 $j \in \{j' : j' = 0, \ldots, r$ 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

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

Algorithm 3. Recursive procedure able to generate the spanning trees of a connected graph $G = (N, E)$ which return a value of the objective function less or equal to Δ

INPUT: graph $G = (N, E)$, integer costs vector c, optimal solution T^*, vector of the non-zero reduced costs $\bar{rc}^* = \{rc_1^*, rc_2^* \dots rc_K^*\}$ (sorted from the lowest to the highest) where K is the number of non-zero reduced costs, the list $FT = \{T^*\}$ containing at the beginning only the initial optimal spanning tree T^*, and the upper bound Δ.

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

1: $it = 1$
2: **while** $(it \leq K)\&\&(obj(T^*) \leq \Delta)$ **do** /*with $obj(T^*)$ value of the objective function in T^**/
3: Let e be the edge not in T^* corresponding to the element $rc_{it}^* \in \bar{rc}$ ($\bar{rc}_1^*$ is the minimum non-zero reduced cost)
4: Let C^* be the cycle that the edge e creates with the edges belonging to the spanning tree T^*
5: Sort by cost (from the highest to the lowest) the edges of the cycle C^* different from e
6: **for** $e^* \in C^*$ with $e^* \neq e$ **do**
7: Add e to T^* and remove e^* from T^*
8: Let $T^{'}$ be the spanning tree so obtained
9: **if** $obj(T^{'}) \leq \Delta$ **then**
10: Add $T^{'}$ to the list FT
11: **if** $it > 1$ **then**
12: Let $rc^{'}$ the vector containing the first $it - 1$ reduced costs
13: **find_feasible_trees** $(G = (N, E), c, T^{'}, rc^{'}, it - 1, FT)$
14: **end if**
15: **end if**
16: **end for**
17: $it = it + 1$
18: **end while**
19:

OUTPUT: Set FT containing the spanning trees of the graph G which return a value of the objective function less or equal to Δ.

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 argument above proves that $\mathcal{E}_i$ contains a complete set of efficient solutions but it may also contain, when the recursion stops, 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 (lines 24–26) checks if $\mathcal{E}_i$ contains any 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. □

5.3. Illustrative example

In this section, we illustrate the computation of the **Two_phase_btree** algorithm on the small undirected graph represented in Fig. 3 with six nodes, seven edges $E = \{e_1 = (1, 2), e_2 = (1, 4), e_3 = (2, 3), e_4 = (2, 5), e_5 = (3, 6), e_6 = (4, 5), e_7 = (5, 6)\}$, and edge costs $c_1 = [29, 25, 13, 17, 20, 35, 10]$ and $c_2 = [15, 25, 20, 17, 13, 10, 35]$.

Table 1 reports the three extreme efficient solutions generated in the first phase by Bensolve, which create the two triangles T_1 and T_2 shown in Fig. 4.

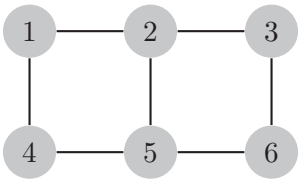


Fig. 3. Example graph.

Table 1
Extreme efficient solutions computed by Bensolve

Tree	Point
{(1, 2), (1, 4), (2, 3), (2, 5), (5, 6)}	(94, 112)
{(1, 2), (1, 4), (2, 3), (2, 5), (3, 6)}	(104,90)
{(1, 2), (2, 3), (2, 5), (3, 6), (4, 5)}	(114, 75)

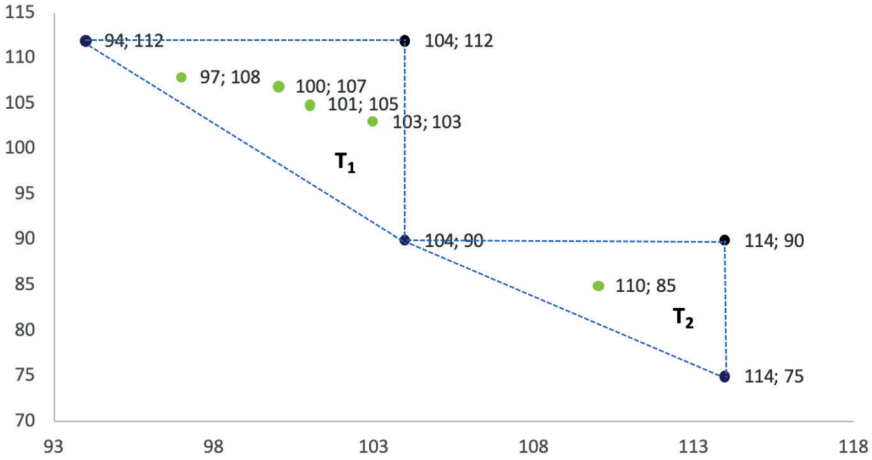


Fig. 4. Pareto Frontier of the illustrative example.

Table 2 reports the solutions in the order in which they were generated by the **Two_phase_btree** algorithm in the first triangle T_1 obtained by the first pair of extreme efficient solutions.

From Table 2 we can observe that the recursive procedure applied in triangle T_1 generates 13 spanning trees (3 of them are generated twice), which correspond to 5 points belonging to triangle T_1, 7 points belonging to triangle T_2 and 1 point outside of both triangles. Among the 5 points inside triangle T_1, 4 of them are non-dominated and thus they correspond to non-supported efficient spanning trees. The related Pareto frontier is shown in Fig. 4.

6. Experimental results

The two-phase strategies TP #1 (PolySCIP), TP #2 (Bensolve) and TP #3 (Prim's algorithm embedded in a weighted sum approach), whose first phases were illustrated in Sections 4.1 and 4.2,

Table 2
Solutions computed by **Two_phase_btree** for triangle T_1 of the illustrative example

#	Tree	Weighted cost	Point	Comment
1	{(1, 2), (1, 4), (2, 3), (3, 6), (5, 6)}	3214	(97, 108)	Non-dominated
2	{(1, 2), (1, 4), (2, 5), (3, 6), (5, 6)}	3272	(101,105)	Non-dominated
3	{(1, 2), (2, 3), (2, 5), (4, 5), (5, 6)}	3258	(104, 97)	Dominated
4	{(1, 4), (2, 3), (2, 5), (4, 5), (5, 6)}	3270	(100, 107)	Non-dominated
5	{(1, 2), (1, 4), (2, 3), (4, 5), (5, 6)}	3514	(112 105)	not in T_1 (and T_2)
6	{(1, 2), (2, 3), (3, 6), (4, 5), (5, 6)}	3284	(107,93)	not in T_1 (and T_2)
7	{(1, 4), (2, 3), (3, 6), (4, 5), (5, 6)}	3296	(103, 103)	Non-dominated
8	{(1, 2), (1, 4), (2, 3), (3, 6), (4, 5)}	3514	(122,83)	not in T_1 (and T_2)
9	{(1, 2), (1, 4), (2, 3), (4, 5), (5, 6)}	3294	(112,105)	not in T_1 (and T_2)
10	{(1, 2), (1, 4), (3, 6), (4, 5), (5, 6)}	3598	(119, 98)	not in T_1 (and T_2)
11	{(1, 2), (2, 5), (3, 6), (4, 5), (5, 6)}	3342	(111, 90)	not in T_1
12	{(1, 4), (2, 5), (3, 6), (4, 5), (5, 6)}	3354	(107, 100)	not in T_1 (and T_2)
13	{(1, 2), (1, 4), (3, 6), (4, 5), (5, 6)}	3598	(119, 98)	not in T_1 (and T_2)
14	{(1, 2), (2, 3), (2, 5), (3, 6), (4, 5)}	3258	(114,75)	not in T_1
15	{(1, 4), (2, 3), (2, 5), (3, 6), (4, 5)}	3270	(110, 85)	not in T_1
16	{(1, 2), (1, 4), (2, 3), (3, 6), (4, 5)}	3514	(122, 83)	not in T_1 (and T_2)

respectively, have been 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 topologies adopting the procedure described in Steiner and Radzik (2008), considering different numbers of vertices. With respect to the grid graphs, we generated instances of 4, 9, 16, 25, 36, 49, 64, 81, 100 and 121 nodes while for the class of complete graphs we generated instances of 10, 14, 18, 22, 26, 34 and 38 nodes. For each set, we considered weak and strong correlations between edge costs. More in detail, we developed a procedure to generate 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 $\alpha = [rand_1 * 2\delta + (45^\circ) - \delta)]$ 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 into an angle α between $45^\circ - \delta$ and $45^\circ + \delta$ 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 $\alpha = 45^\circ$ 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 positive correlation and to weaker negative correlation).

For each set of parameters, we generated 10 graphs. We set a time limit of 600 s of CPU time for each run and we report the average running time of the 10 runs distinguishing between the time devoted to the first and the second phase. Moreover, we run TP #1, TP #2 and TP #3 also on the instances by Di Puglia Pugliese et al. (2018) available at http://www.mat.uc.pt/~zeluis/INVESTIG/MOMST/momst.htm. This section reports average results of our computational experiments and the comparison with the ones related to the most recent two-phase strategy by Steiner and Radzik (2008) and the most recent algorithm by Di Puglia Pugliese et al. (2018) designed for the BMST problem. Detailed results, instance by instance, can be found in the Appendix. Note that the considered problem sizes are not comparable with the ones used in Sourd and Spanjaard (2008). Indeed,

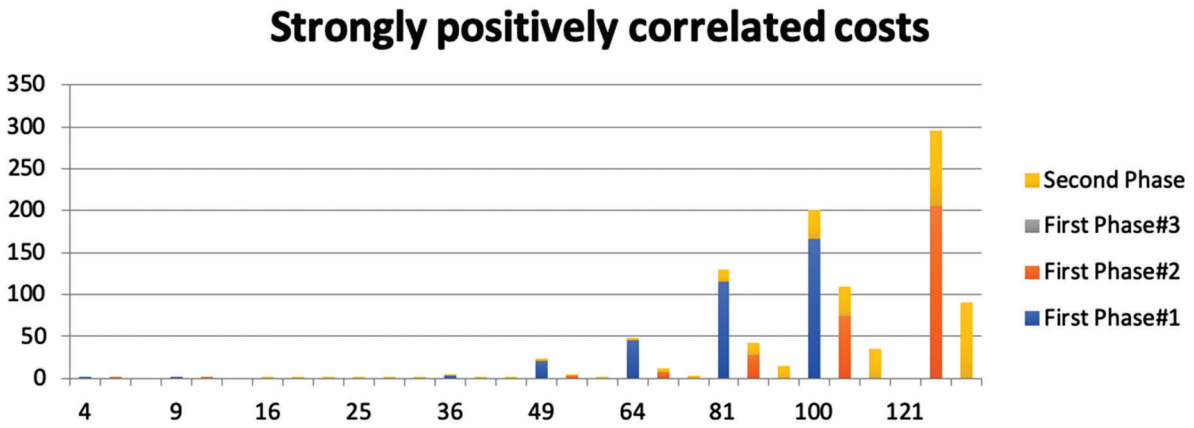


Fig. 5. Average solution time (s) for grid graphs with strongly positively correlated costs.

an important difference between the algorithm presented in this paper and the branch-and-bound approach proposed by Sourd and Spanjaard is that the first one leads to compute the complete set of efficient solutions X_E, i.e., a complete set of efficient spanning trees in the decision space, whereas the latter one computes Y_N, the set of non-dominated outcomes in the image space. This implies that $|X_E|$ is generally exponential while $|Y_N|$ is pseudopolynomial since it can contain at most nK points, where K is the maximum edge weight. Therefore, any approach that searches for Pareto trees is necessarily limited to smaller instances because computation time is in $(|X_E|)$, although it has the advantage of providing the actual efficient trees rather than only their objective function values.

First phase 1 was developed using PolySCIP (Schenker and Strunk, 2016), which 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, see Borndörfer et al. (2016). The implementation of the First phase 2 is based on Bensolve, see Weißing and Löhne (2014), which is an open-source solver for vector linear programs and, in particular, for multiple objective linear programs based on Benson's algorithm (Weißing and Löhne, 2017). More precisely, we used this solver resorting to the dual variant of Benson's algorithm (Ehrgott et al., 2012). The First phase #3, consisting in Prim's algorithm embedded in a weighted sum method by means of a recursive interval based algorithm, and the recursive procedure on which the second phase is based, have been implemented by the authors in C++. The whole code has been developed in Xcode for Mac and all tests were run on a node of a high-performance computing cluster, composed of four nodes, each of them with 32 cores and 64 GB of RAM.

In Fig. 5 the computational times related to grid graphs with strongly positively correlated costs are reported. Similarly, in Fig. 7, we show the results of grid graphs with weakly positively correlated edge costs. For each set of instances, we distinguished the running times of the two phases, considering the three different strategies for the first phase (First phase 1, First phase 2, First phase 3).

From Fig. 5 we can observe that all instances can be solved within the time limit (600 s). First phase 3 is the fastest one in all cases, followed by First phase 2 and First phase 1 (note that for the set of grid graphs with 121 nodes PolySCIP is able to solve only one instance within 600 CPU seconds

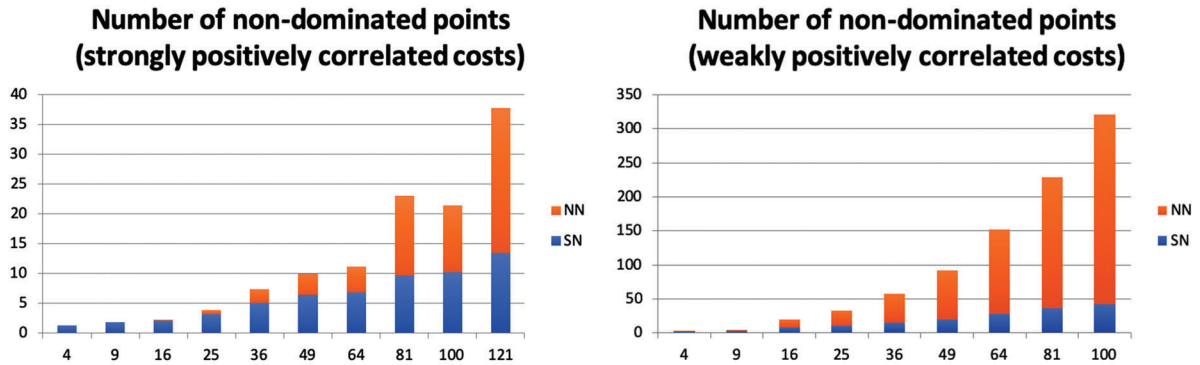


Fig. 6. Average number of non-dominated points for grid graphs with positively correlated costs.

and for this reason it does not appear in the histogram). Consequently, the total running time of the two-phase method is smaller using Prim's algorithm for the first phase. This is an expected behaviour, as the latter is a greedy algorithm. However, these results show that it is also possible to solve the problems up to the maximum considered size in less than 5 minutes, at least for strong positive correlation instances, by means of a mathematical programming approach (dual variant of Benson algorithm). The average solution time of the second phase ranges between 0 and 89 CPU seconds. Thus, considering the fastest strategy TP #3, we can observe that for all instances belonging to this class, the first phase needs a running time that is almost always equal to 0 and the second phase takes up most of the solution time which increases with the input size. This behaviour can be understood by looking at the number of non-dominated points.

Figure 6(a) shows the average number of non-dominated points for each set of instances, distinguishing between supported non-dominated points (SN) and non-supported non-dominated points (NN). We can see that the number of extreme non-dominated points augments with the input size and thus the number of triangles that have to be analysed in the second phase grows. For this reason the running time of the second phase increases. Moreover, we note that for instances with 4 and 9 nodes there are no non-supported non-dominated (NN) points in the Pareto frontier, while from instances of 16 nodes or more we have an increasing number of non-supported non-dominated points. For instances with 81, 100 and 121 nodes, they represent the majority of the non-dominated points. This is also an expected behaviour because as already observed the number of non-supported non-dominated points increases with the input size. Similar observations can be made for weakly positively correlated costs from Fig. 7, but for this class of instances, our two-phase algorithm is not able to solve problems with 121 nodes within the time limit.

In the case of weakly positive correlation, we detect from Fig. 6(b) that for each size of instances the average number of non-supported non-dominated points is larger than the mean number of supported ones and they appear in the Pareto frontier also for instances of 4 and 9 nodes. This is due to the weak correlation between costs which leads to enlarging the angle formed by the gradients of the objective functions, that is, to more conflict between objectives and thus to an increase of the cardinality of the Pareto frontier. For this class of instances, Table 3(a) reports also the comparison between the running times of the two-phase algorithm TP#3 and the ones provided

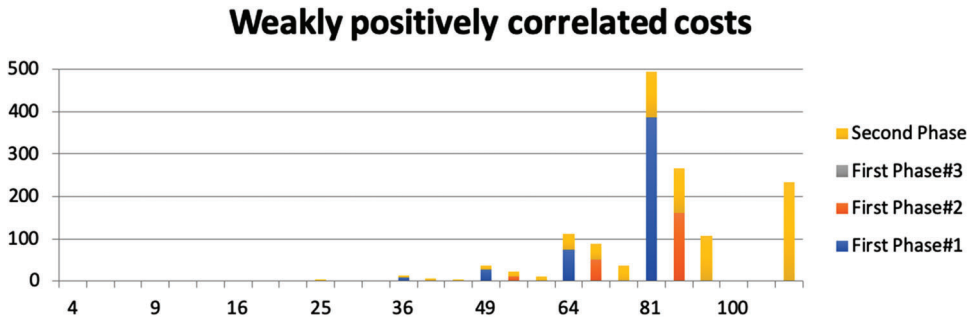


Fig. 7. Average solution time (s) for grid graphs with weakly positively correlated costs.

Table 3
Running time comparison with Stainer and Radzik (S&R) algorithm

(a) Grid graphs (weakly positively correlated costs)			(b) Complete graphs (strongly positively correlated costs)		
Running times (in seconds) for grid graphs with weakly positively correlated costs			Running times (in seconds) for complete graphs with strongly positively correlated costs		
n	S&R	TP#3	n	S&R	TP#3
4	0	0	10	0.01	0.03
9	0	0	14	0.04	0.11
16	0.06	0.14	18	0.26	0.82
25	0.4	0.84	22	1.65	1.23
36	1.35	3.59	26	3.37	0.46
49	5.14	10.8	30	9.76	0.14
64	18.5	37.92	34	24.32	0.17
81	58.27	107.97	38	65.64	0.46
100	166.8	233.24			
121	328.26	–			

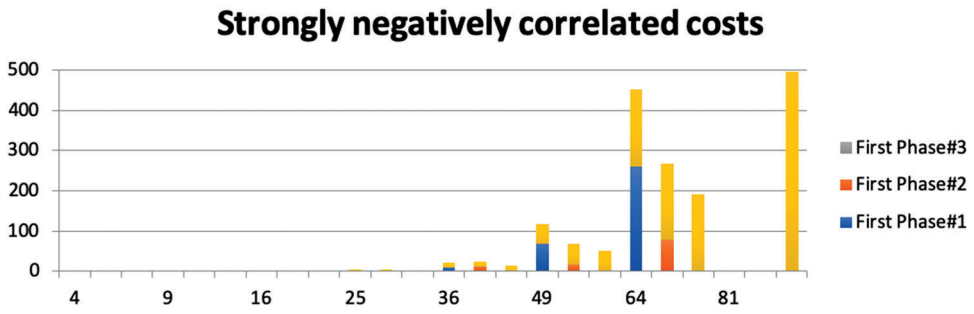
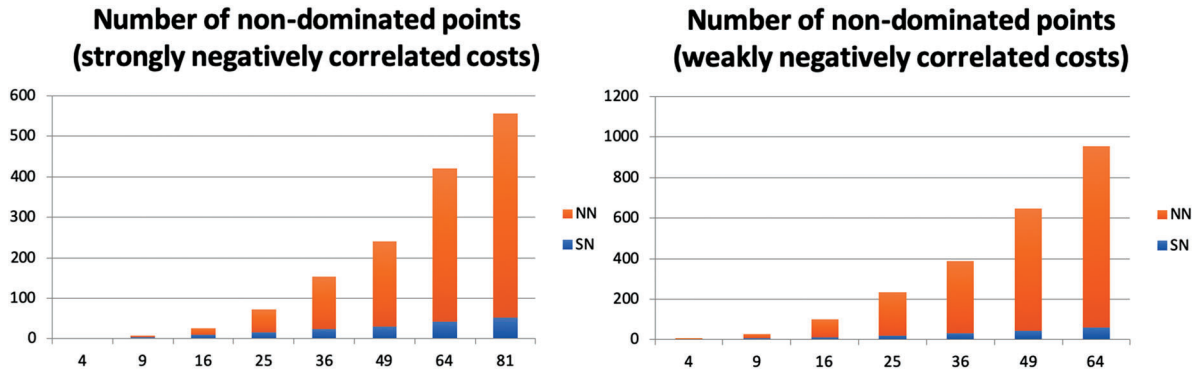


Fig. 8. Average solution time (s) for grid graphs with strongly negatively correlated costs.

by Steiner and Radzik (2008) (S&R). They show that for this set of instances Steiner and Radzik method is faster.

We also considered negative correlation between costs, always distinguishing between strong and weak correlation. Figure 8 reports the average solution times of the two-phase algorithm for



(a) strongly negatively correlated costs (b) weakly negatively correlated costs

Fig. 9. Average number of non-dominated points for grid graphs with negatively correlated costs.

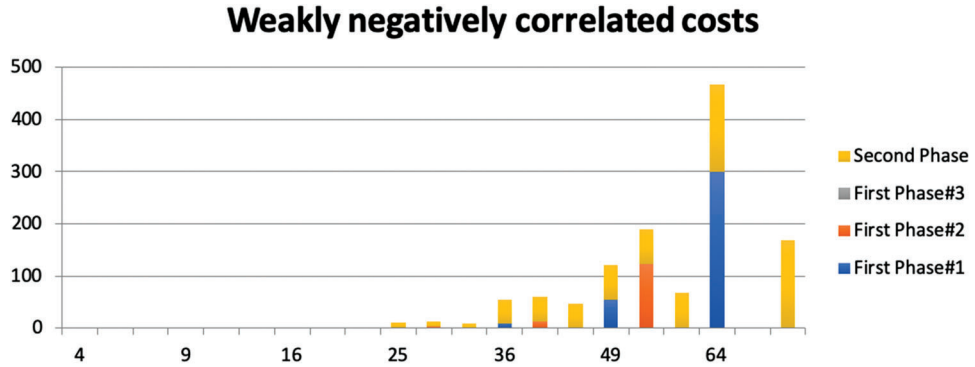


Fig. 10. Average solution time (s) for grid graphs with weakly negatively correlated costs.

strongly negatively correlated costs. In this case, the fastest version of our method (TP#3) is able to solve grid instances up to 81 nodes within the time limit. Indeed, the size of the Pareto frontier is further increased due to the negative correlation between costs as can be observed in Fig. 9(a). The number of non-dominated points of each set of instances is more than double that in the case of weakly positively correlated costs.

Regarding the instances with weakly negatively correlated costs, Fig. 10 shows that, even if First phase 3 is always the fastest one, First phase 1 performs significantly better than First phase 2, differently from the previous cases. Indeed, the two-phase algorithm is able to solve instances of 64 nodes within the time limit applying PolySCIP or Prim's algorithm embedded in the weighted sum method. The size of the Pareto frontier further increases and the supported non-dominated points represent a very small proportion of it, as shown in Fig. 9(b).

Similarly, we tested the two-phase algorithm on complete graphs from 10 to 38 nodes like in Steiner and Radzik (2008). In Fig. 11, we can see that the algorithm is able to generate the Pareto frontiers within the time limit for each set of instances when the costs are strongly positively correlated. In this case, from instances of size 30, First phase 1 performs better than First phase 2 but the fastest one remains, as expected, First phase 3. The second phase takes a very limited amount of the

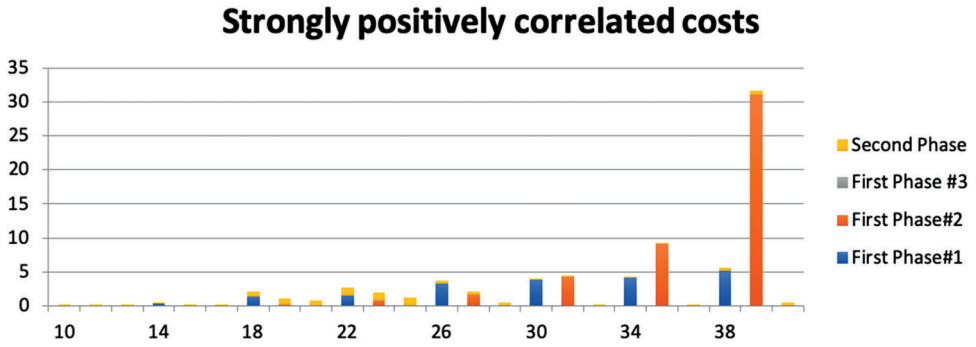


Fig. 11. Average solution time (s) for complete graphs with strongly positively correlated costs.

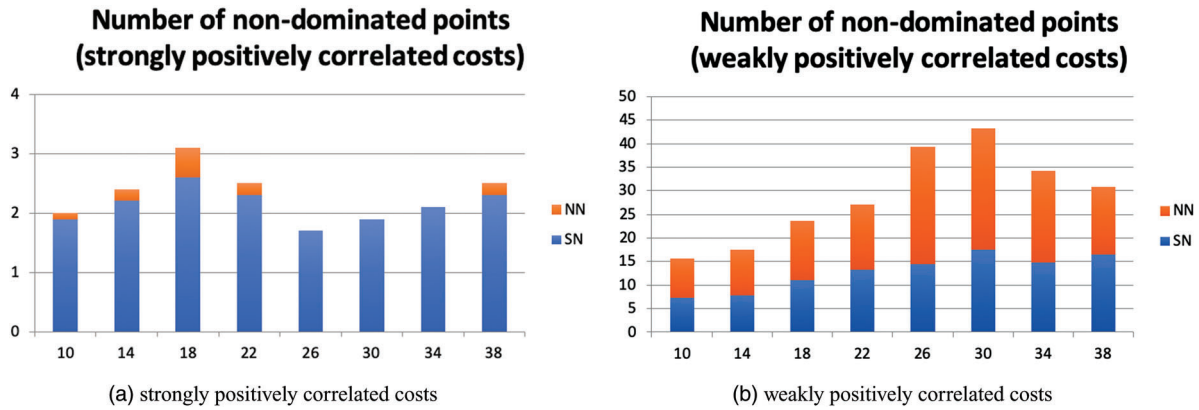


Fig. 12. Average number of non-dominated points for complete graphs with positively correlated costs.

total running time. Indeed, the average number of non-dominated points is always around 2 with the exception of the set of instances with 18 nodes for which it is roughly speaking 3, Fig. 12(a). The limited number of efficient solutions can be explained considering that the strong positive correlation between costs originated test instances characterised by a large domination cone.

Table 3(b) reports also the comparison between the running times of the two-phase algorithm TP#3 and the ones provided by Steiner and Radzik (2008) (S&R) for complete graphs with strongly positively correlated costs. In this case, we can observe better performance of our method, with a maximum saving in running time of 99% for complete graphs with 38 nodes.

Also in the case of weak positive correlation, the algorithm is able to generate a complete set of efficient solutions within the time limit for all the instances (Fig. 13). The second phase is slower than the first one for each set. Indeed, as shown in Fig. 12(b), the number of non-dominated extreme points is bigger than in the case of a strong positive correlation. Moreover, the total number of non-dominated points increases with respect to the case of strong correlation, since the weak correlation between costs implies a smaller domination cone. Moreover, the number of non-supported non-dominated points increases with the size of the instances up to the set of complete graphs with 30 nodes (Fig. 12b).

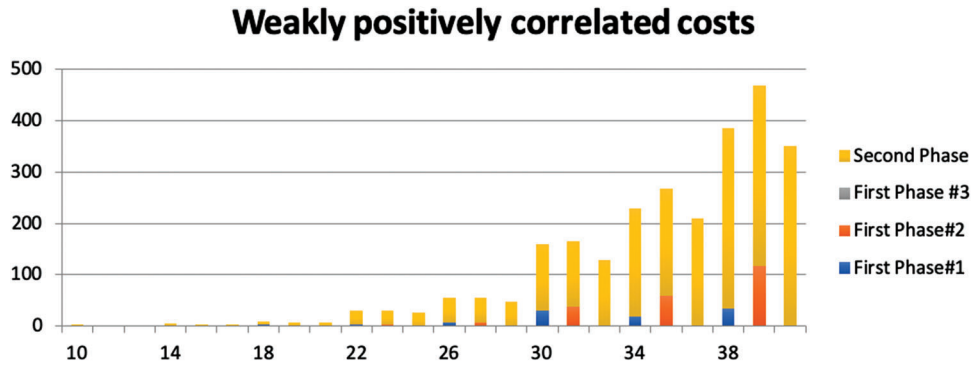


Fig. 13. Average solution time (s) for complete graphs with weakly positively correlated costs.

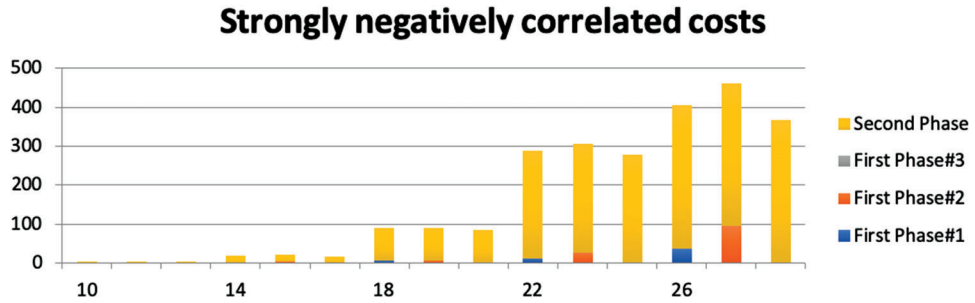


Fig. 14. Average solution time (s) for complete graphs with strongly negatively correlated costs.

Considering negative correlation, similarly to the class of grid graphs, the size of the instances for which the algorithm is able to generate the Pareto frontier within the time limit decreases. As we can observe from Fig. 14, the maximum solvable size is 26 nodes. First phase 3 continues to be the best strategy and the second phase takes up most of the solution time because complete graphs imply a higher number of fundamental cycles to be examined through the recursive procedure. Figure 15(a) shows that the set of non-dominated points is mainly composed of non-supported ones.

The weak negative correlation between costs makes the instances harder to solve and, as represented in Fig. 16, the maximum solvable size, within the time limit, decreases to 18 nodes.

The comparison between the three strategies for the first phase appears more similar than in the previous cases even if First phase 1 continues to be faster than the First phase 2. First phase 3 is the fastest one. The number of non-supported non-dominated points always increases with the size of the instances and they represent almost the entire Pareto frontier, Fig. 15(b).

Table 4 reports the comparison between the running times of the two-phase algorithm TP#3 and the ones provided by Di Puglia Pugliese et al. (2018) (BN) on their instances. Di Puglia Pugliese algorithm is faster than ours although the larger the size the smaller the difference in the running times between their and our algorithm. This trend seems to indicate that further increasing the size of the instances would yield the performance of both algorithms to be similar. Nevertheless, we

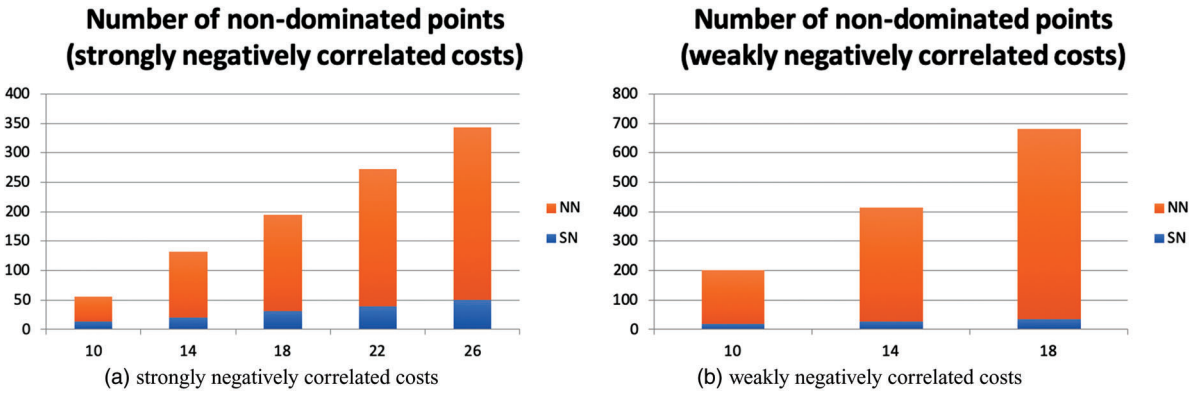


Fig. 15. Average number of non-dominated points for complete graphs with negatively correlated costs.

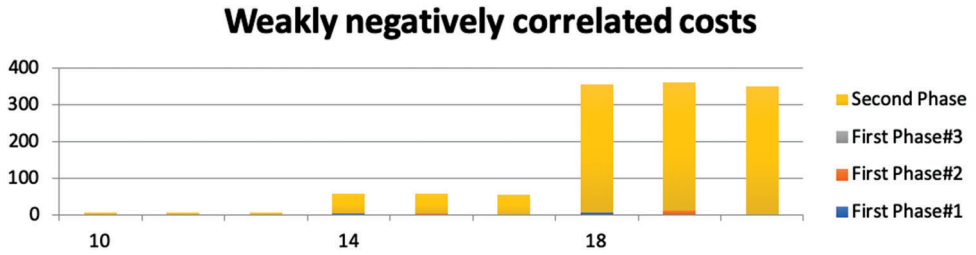


Fig. 16. Average solution time (s) for complete graphs with strongly negatively correlated costs.

Table 4
Running time comparison with Di Puglia Pugliese et al. algorithm

Running times (in ms) for Di Puglia Pugliese et al. instances		
n	BN	TP#3
5	0.18	7.07
6	0.20	23.73
7	0.61	59.37
8	1.58	135.89
9	3.26	211.01
10	17.54	452.61
11	41.08	698.86
12	110.91	1156.21
13	344.52	1691.21
14	1122.22	2586.84

could not confirm our guess on this behaviour of the algorithms for larger sizes since the dataset provided by Di Puglia Pugliese et al. (2018) only goes until 14 nodes.

Summing up, from our computational experience, it appears that Bensolve outperforms PolySCIP for grid graph instances with the exception of weakly negatively correlated costs. PolySCIP compares favourably with Bensolve for complete graph instances. This behaviour is

possibly due to the different formulations adopted. Indeed, the Kipp-Martin formulation that must be used with Bensolve is characterized by a larger number of variables with respect to the flow formulation used with Polyscip. This difference makes the Kipp-Martin formulation more complex to be solved when the graphs are complete and thus the number of variables z_{kij} increases considerably. As expected, in all cases the first phase based on Prim's algorithm outperforms the others. However, the adoption of a mathematical programming approach is reasonable up to instances of size 36 for grid graphs with positively correlated costs, for which the running time of Bensolve, even if it is longer than the one associated with Prim's algorithm, is still limited. Also for complete graphs with positively correlated costs, the mathematical programming approach can be adopted without very big differences in terms of running time up to size 18, resorting to Polyscip.

As far as the second phase is concerned, the solution time is correlated to the average number of non-dominated extreme points of the convex hull defined by the feasible region in the objective space. As could be expected, negatively correlated costs give rise to much larger non-dominated sets, also including a larger proportion of non-supported non-dominated points. This structure makes it more difficult to compute the sets of Pareto solutions increasing significantly the requirements on computing time. Moreover, we compared our algorithm with the most recent two-phase method, Steiner and Radzik (2008) and with the most recent algorithm designed for the multi-objective MST problem Di Puglia Pugliese et al. (2018).

Concerning the comparison in the case of Steiner and Radzik (2008), our algorithm performs better in the class of complete graphs with strongly positively correlated costs and their method performs better for grid graphs. As regards the comparison with Di Puglia Pugliese et al. (2018), we observed their algorithm performs better than ours, but as the size of the problems increases the behaviour becomes similar. Our results confirm what is already known in the literature about the difficulty of these problems related with the graph topology and edge cost correlations. The proposed method is not always faster than already existing ones although it solves similar instance sizes. This behaviour can be explained by observing that the enumerative recursive procedure does not find feasible solutions in strict ranking order. Consequently, the update of the upper bound, which limits the recursion, can be in some cases slower, making the whole second phase not always faster than the one of previous algorithms proposed in the literature. Thus, at least in our implementation, the more general applicability of the procedure, in some cases, results in a convergence of the algorithm not faster than the already existing ones.

7. Conclusions

This paper presents a new two-phase algorithm to compute a complete set of Pareto solutions of the BMST problem. In the first phase, the extreme efficient solutions are obtained by three different strategies: (1) a weighted sum approach that can be implemented by any mixed-integer linear programming solver although we have applied it with PolyScip; (2) a dual variant of Benson's algorithm that is applied to the Kipp-Martin formulation of the BMST problem and (3) Prim's greedy algorithm embedded in a weighted sum approach. The second phase is based on a new recursive algorithm which takes advantage of the particular structure of the problem and its basic feasible solutions able to generate all the spanning trees of a connected network. Extensive computational experiments show the correctness of our new approach. We solve problems on two different graph

topologies (grid and complete) and with different cost structures (positively and negatively correlated). Moreover, we also compare our algorithm with the most recent two-phase method (Steiner and Radzik, 2008) and with the most recent algorithm designed for the multi-objective MST problem (Di Puglia Pugliese et al., 2018). Qualitative analysis of the solutions based on the graph topology and cost structure are reported. All the results are included in the electronic appendix associated with this paper. Although the proposed approach is not always faster than the existing ones, it solves problem instances of a size similar to those already considered in the literature. Moreover, the presented recursive procedure adopted in the second phase, has the advantage to be easy to implement and the potential to be embedded in a two-phase method to deal with a wider class of combinatorial optimization problems. Indeed, it exploits the spanning tree structure of the basic feasible solutions, which is a property satisfied by single- and multi-commodity network flow problems (e.g., min cost flow problem and shortest path problem). This opens a new line of research in designing two-phase methods not relying on the adoption of "ad hoc" procedures for problems belonging to this class.

Acknowledgements

This paper originated during a three-month stay of Lavinia Amorosi at the University of Seville supported by Sapienza University of Rome via Ph.D. students mobility grant no. 4389/2016. Further research to complete it has also been supported by Sapienza University of Rome via project no. AR11715C587D77BB. The second author research has been partially supported by the Agencia Estatal de Investigación (AEI) and the European Regional Development's funds (ERDF): PID2020-114594GB-C21; Regional Government of Andalusia: projects CEI-3-FQM331, FEDER-US-1256951, and P18-FR-1422; Fundación BBVA: project NetmeetData (Ayudas Fundación BBVA a equipos de investigación científica 2019).

Open Access Funding provided by Universita degli Studi di Roma La Sapienza within the CRUI-CARE Agreement.

References

Amaruchkul, K., 2021. Multiobjective land-water allocation model for sustainable agriculture with predictive stochastic yield response. International Transactions in Operational Research. https://onlinelibrary.wiley.com/doi/pdf/10.1111/itor.13015.

Amorosi, L., 2018. Bi-criteria network optimization: problems and algorithms. Ph.D. thesis, Sapienza University of Rome.

Amorosi, L., Ehrgott, M., 2018. A new two-phase algorithm for the bi-objective integer min cost flow problem. Working paper.

Benson, H.P., 1998. 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* 13, 1–24.

Bissoli, D.C., Zufferey, N., Amaral, A.R.S., 2021. Lexicographic optimization-based clustering search metaheuristic for the multiobjective flexible job shop scheduling problem. *International Transactions in Operational Research* 28, 5, 2733–2758.

Borndörfer, R., Schenker, S., Skutella, M., Strunk, T., 2016. Polyscip. *Mathematical Software – ICMS 2016, 5th International Conference, Berlin, Germany, July 11-14, 2016, Proceedings* 9725, 259–264.

Corley, H.W., 1985. Efficient spanning trees. *Journal of Optimization Theory and Applications* 45, 481–485.

Di Puglia Pugliese, L., Guerriero, F., Santos, J., 2015. Dynamic programming for spanning tree problems: application to the multi-objective case. *Optimization Letters* 9, 3, 437–450.

Di Puglia Pugliese, L., Guerriero, F., Santos, J., 2018. A new approach for the multiobjective minimum spanning tree. *Computers and Operations Research* 98, 69–83.

Ehrgott, M., 2005. *Multicriteria Optimization*. Springer, Berlin.

Ehrgott, M., 2006. A discussion of scalarization techniques for multiple objective integer programming. *Annals of Operations Research* 147, 343–360.

Ehrgott, M., Gandibleux, X., 2000. A survey and annotated bibliography on multiobjective combinatorial optimization. *OR Spektrum* 22, 425–460.

Ehrgott, M., Klamroth, K., 1997. Connectedness of efficient solutions in multiple criteria combinatorial optimization. *European Journal of Operational Research* 97, 159–166.

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

Erdoğdu, K., Karabulut, K., 2021. Bi-objective green vehicle routing problem. *International Transactions in Operational Research* 29, 3, 1602–1626.

Fernández, E., Pozo, M.A., Puerto, J., Scozzari, A., 2017. Ordered weighted average optimization in multiobjective spanning tree problem. *European Journal of Operational Research* 260, 3, 886–903.

Gabow, H., 1977. Two algorithms for generating weighted spanning trees in order. *Journal on Computing* 6, 139–150.

Gorski, J., Klamroth, K., Ruzika, S., 2011. Connectedness of efficient solutions in multiple objective combinatorial optimization. *Journal of Optimization Theory and Applications* 150, 475–497.

Gross, J., Yellen, J., 2006. *Graph Theory and Its Applications*. Taylor & Francis Group, London.

Hamacher, H., Ruhe, G., 1994. On spanning tree problems with multiple objectives. *Annals of Operations Research* 52, 209–230.

Heyde, F., Löhne, A., 2008. Geometric duality in multiple objective linear programming. *Journal of Optimization* 12, 836–845.

Kipp-Martin, R., 1991. Using separation algorithms to generate mixed integer model reformulations. *Operations Research Letters* 10, 119–128.

Knowles, J.D., Corne, D.W., 2002. 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* 143, 543–547.

Leitner, M., Ljubic, I., Sinnl, M., 2015. The bi-objective prize-collecting steiner tree problem. *INFORMS Journal on Computing* 27, 1, 118–134.

Magnanti, T., Wolsey, L.A., 1995. Optimal trees. *Handbooks in Operations Research and Management Science* 7, 503–616.

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

Prins, C., Prodhon, C., Wolfler Calvo, R., 2006. Two-phase method and Lagrangian relaxation to solve the bi-objective set covering problem. *Annals of Operations Research* 147, 1, 23–41.

Przybylski, A., Gandibleux, X., Ehrgott, M., 2008. Two phase algorithms for the bi-objective assignment problem. *European Journal of Operational Research* 185, 2, 509–533.

Przybylski, A., Gandibleux, X., Ehrgott, M., 2010a. A recursive algorithm for finding all nondominated extreme points in the outcome set of a multiobjective integer programme. *INFORMS Journal on Computing* 22, 3, 371–386.

Przybylski, A., Gandibleux, X., Ehrgott, M., 2010b. A two phase method for multi-objective integer programming and its application to the assignment problem with three objectives. *Discrete Optimization* 7, 3, 149–165.

Queiroz, T.A.d., Mundim, L.R., 2020. Multiobjective pseudo-variable neighborhood descent for a bicriteria parallel machine scheduling problem with setup time. *International Transactions in Operational Research* 27, 3, 1478–1500.

Raith, A., Ehrgott, M., 2009a. A comparison of solution strategies for biobjective shortest path problems. *Computers and Operations Research* 36, 1299–1331.

Raith, A., Ehrgott, M., 2009b. A two-phase algorithm for the biobjective integer minimum cost flow problem. *Computers and Operations Research* 36, 1945–1954.

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

Salamirad, A., Kheybari, S., Ishizaka, A. & Farazmand, H. 2021. Wastewater treatment technology selection using a hybrid multicriteria decision-making method. International Transactions in Operational Research https://onlinelibrary.wiley.com/doi/pdf/10.1111/itor.12979.

Schenker, S., Strunk, T., 2016. Polyscip.

Serafini, P., 1987. 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*. Springer, Berlin, Vol. 294, pp. 222–232.

Sourd, F., Spanjaard, O., 2008. A multiobjective branch-and-bound framework: Application to the biobjective spanning tree problem. *INFORMS Journal on Computing* 20, 3, 333–498.

Steiner, S., Radzik, T., 2008. Computing all efficient solutions of the biobjective minimum spanning tree problem. *Computers and Operations Research* 35, 198–211.

Ulungu, E., Teghem, J., 1994a. Application of the two phases method to solve the bi- objective knapsack problem. Technical Report, Faculté Polytechnique de Mons, Belgium.

Ulungu, E., Teghem, J., 1994b. The two phases method: an efficient procedure to solve bi-objective combinatorial optimization problems. *Foundations of Computing and Decision Sciences* 20, 149–165.

Visée, M., Teghem, J., Pirlot, M., Ulungu, E., 1998. Two-phases method and branch and bound procedures to solve the bi-objective knapsack problem. *Journal of Global Optimization* 12, 139–155.

Weißing, B., Löhne, A., 2014. Bensolve.

Weißing, B., Löhne, A., 2017. The vector linear program solver Bensolve – notes on theoretical background. *European Journal of Operational Research* 260, 807–813.

Xiang, T., Li, Y. & Szeto, W.Y. 2021. The daily routing and scheduling problem of home health care: based on costs and participants preference satisfaction. International Transactions in Operational Research. https://onlinelibrary.wiley.com/doi/pdf/10.1111/itor.13043.

Zhou, G., Gen, M., 1999. Genetic algorithm approach on multi-criteria minimum spanning tree problem. *European Journal of Operational Research* 114, 141–152.