SAPIENZA
UNIVERSITÀ DI ROMA

Bi-criteria Network Optimization: Problems and Algorithms

Scuola di dottorato

Dottorato di Ricerca in Ricerca Operativa – XXX Ciclo

Candidate

Lavinia Amorosi
ID number 1200445

Thesis Advisor

Prof. Paolo Dell'Olmo

A thesis submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Operational Research

2017

Thesis defended on 2018
in front of a Board of Examiners composed by:

Prof. Raffaele Pesenti (chairman)

Prof. Justo Puerto

Bi-criteria Network Optimization: Problems and Algorithms
Ph.D. thesis. Sapienza – University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Version: April 6, 2018

Author's email: lavinia.amorosi@uniroma1.it

Dedicated to
my parents Marcello and Angela

Abstract

Several approaches, exact and heuristics, have been designed in order to generate the Pareto frontier for multi-objective combinatorial optimization problems. Although several classes of standard optimization models have been studied in their multi-objective version, there still exists a big gap between the solution techniques and the complexity of the mathematical models that derive from the most recent real world applications. In this thesis such aspect is highlighted with reference to a specific application field, the telecommunication sector, where several emerging optimization problems are characterized by a multi-objective nature. The study of some of these problems, analyzed and solved in the thesis, has been the starting point for an assessment of the state of the art in multicriteria optimization with particular focus on multi-objective integer linear programming. A general two-phase approach for bi-criteria integer network flow problems has been proposed and applied to the bi-objective integer minimum cost flow and the bi-objective minimum spanning tree problem. For both of them the two-phase approach has been designed and tested to generate a complete set of efficient solutions. This procedure, with appropriate changes according to the specific problem, could be applied on other bi-objective integer network flow problems. In this perspective, this work can be seen as a first attempt in the direction of closing the gap between the complex models associated with the most recent real world applications and the methodologies to deal with multi-objective programming. The thesis is structured in the following way: Chapter 1 reports some preliminary concepts on graph and networks and a short overview of the main network flow problems; in Chapter 2 some emerging optimization problems are described, mathematically formalized and solved, underling their multi-objective nature. Chapter 3 presents the state of the art on multicriteria optimization. Chapter 4 describes the general idea of the solution algorithm proposed in this work for bi-objective integer network flow problems. Chapter 5 is focused on the bi-objective integer minimum cost flow problem and on the adaptation of the procedure proposed in Chapter 4 on such a problem. Analogously, Chapter 6 describes the application of the same approach on the bi-objective minimum spanning tree problem. Summing up, the general scheme appears to adapt very well to both problems and can be easily implemented. For the bi-objective integer minimum cost flow problem, the numerical tests performed on a selection of test instances, taken from the literature, permit to verify that the algorithm finds a complete set of efficient solutions. For the bi-objective minimum spanning tree problem, we solved a numerical example using two alternative methods for the first phase, confirming the practicability of the approach.

Highlights

- New mathematical models for emerging problems in Telecommunications sector and solution analysis; evidence of their multi-objective nature.
- A new two-phase algorithm for bi-objective integer network flow problems based on a recursive procedure for the second phase.
- Bi-criteria minimum integer cost flow problem and Bi-criteria Minimum Spanning Tree Problem solved by the new two-phase algorithm.
- Formal proofs of the completeness of the Pareto frontier generated.
- Generality of the recursive procedure proposed for enumerating the feasible solutions in the second phase, differently from the most recent works [70] and [77] which use ad hoc algorithms for the minimum integer cost flow problem and the minimum spanning tree problem respectively.
- Experimental results on both problems that show the applicability of the approach.

Acknowledgments

Firstly, I would like to express my sincere gratitude to my advisor Prof. Paolo Dell'Olmo for the continuous support of my Ph.D study and related research, for his patience, motivation, and knowledge. His guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better advisor and mentor for my Ph.D study.
I wish to thank Prof. Matthias Ehrgott for his ideas and contributions to the two-phase algorithm for the minimum cost flow problem during my staying in Lancaster and Prof. Justo Puerto for his help during my staying in Seville and for his advices and constructive suggestions for the two-phase algorithm for the minimum spanning tree problem. Without their precious support it would not be possible to conduct this research.
My sincere thanks also go to Prof. Luca Chiaraviglio, for introducing me to the emerging optimization problems in Telecommunications and to Prof. Andrea Raith for providing me the test instances of the bi-objective minimum cost flow problem.
I would like to thank also my friend Dr. Massimo Vespignani for reviewing overnight the final version of this thesis from Mountain View. Last but not the least, I would like to thank my family: my parents and my sister for supporting me spiritually throughout writing this thesis and my life in general.

Contents

Introduction

In many real world applications optimization problems arise where it is required to deal simultaneously with more than one criteria. Several examples of such problems, related to different application fields, can be mentioned. In the manufacturing industry, both maximization of the gain and minimization of the pollution associated to the production process have to be considered simultaneously. Analogously in the transport sector energy efficiency and quality of service have to be guaranteed. The same is true in the health-care system, where quality of service and minimization of the costs represent the main decision criteria. In finance, maximization of the profit and minimization of the risk are the primary goals in portfolio selection. Many other examples could be reported, characterized by two or more objectives that usually, as in those mentioned above, are in conflict with each other. Such conflicting relation between criteria, represents the base for the development of the multi-objective programming. Indeed when more than one conflicting goals have to be optimized it is not possible to use the same concept of "best" solution as for a single objective problem. For such a reason a new concept of optimum has been defined, in order to deal with this class of problems. The first definition of optimum in the context of multi-criteria economic decision making was by Francis Y. Edgeworth. Considering two hypothetical consumer criteria, P and π, he asserted that "It is required to find a point (x, y) such that in whatever direction we take an infinitely small step, P and π do not increase together but that, while one increases, the other decreases" (Edgeworth 1881). Pareto was a contemporary of Edgeworth, and in his most famous theory, "The Pareto Optimum", gave his definition of Pareto optimality: "The optimum allocation of the resources of a society is not attained so long as it is possible to make at least one individual better off in his own estimation while keeping others as well off as before in their own estimation" (Pareto 1906). This new definition of optimality implies the existence of more than one Pareto optimal solution (Pareto frontier), equally efficient, for a multi-objective problem. Such solutions are trade-off solutions for the problem and they are incomparable to each other. From the concept of Pareto optimality new techniques were developed for finding the Pareto frontier. As in the single objective case, the complexity of such methodologies increases when integer or binary decisional variables are considered. Several approaches, exact and heuristics, were designed in order to generate the Pareto frontier for multi-objective combinatorial optimization problems. Although several classes of standard optimization models have been studied in their multi-objective version, there still exists a big gap between the solution techniques and the complexity of the mathematical models that derive from the most recent real world applications.

In this thesis such aspect is highlighted with reference to a specific application field, the telecommunication sector, where several emerging optimization problems are characterized by a multi-objective nature. The study of these problems has been the starting point for an assessment of the state of the art in multi-criteria optimization with particular focus on multi-objective integer linear programming. From the current state of the research, two bi-criteria optimization problems, that have important applications also in the telecommunication industry, have been studied: the bi-objective integer minimum cost flow and the bi-objective minimum spanning tree problems. For both of them a two-phase approach has been designed, able to generate a complete set of efficient solutions. This algorithm, with appropriate changes according with the specific problem, could be applied on a generic bi-objective network flow problem. The results represent a first attempt in the direction to close the gap between the complex models associated with the most recent real world applications and the methodologies to deal with multi-objective programming.
The thesis is structured in the following way: Chapter 1 reports some preliminary concepts on graph and networks and a short overview of the main network flow problems; in Chapter 2 some emerging optimization problems are described, mathematically formalized and solved, underling their multi-objective nature. Chapter 3 presents an overview on multicriteria optimization. Chapter 4 describes the general idea of a new solution algorithm proposed for integer network flow problems. Chapter 5 is focused on the bi-objective integer minimum cost flow problem and on the adaptation of the procedure proposed in Chapter 4 on this problem. Analogously, Chapter 6 describes the application of the same approach on the bi-objective minimum spanning tree problem. For the integer min cost flow problem, the numerical tests performed on a selection of test instances, taken from the literature, permit to verify that the algorithm finds a complete set of efficient solutions. For the minimum spanning tree problem, we solved a numerical example using two alternative methods for the first phase, confirming the practicability of the approach.

Chapter 1

Preliminary concepts

In this chapter the main concepts and definitions that will be used in this work are presented. In Section 1.1 several basic definitions and results from graph theory are recalled (taken from [3], [43] and [65]). In section 1.2, an overview of the main network flow problems and their applications is given (taken from [3] and [64]). Lastly Section 1.3 contains a short description of the network simplex method (from [3]).

1.1 Graphs and related definitions

Definition 1 (Directed Graphs)**.** *A directed graph $G = (N, A)$ consists of a set N of nodes and a set A of arcs whose elements are ordered pairs of distinct nodes. A directed network is a directed graph whose nodes and/or arcs have associated numerical values (such as costs, capacities, and/or supplies and demands).*

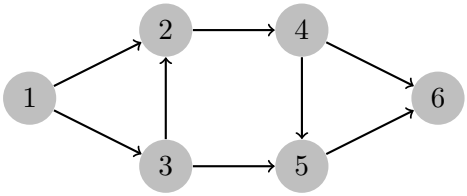


Figure 1.1. Example of directed graph

Definition 2 (Undirected Graphs)**.** *An undirected graph $G = (N, A)$ consists of a set N of nodes and a set A of arcs whose elements are unordered pairs of distinct nodes.*

Definition 3 (Tails and Heads)**.** *A directed arc (i, j) has two endpoints i and j. The node i is the tail $t(i, j)$ of arc (i, j) and node j is its head $h(i, j)$. The arc (i, j) is an outgoing arc of node i and an incoming arc of node j.*

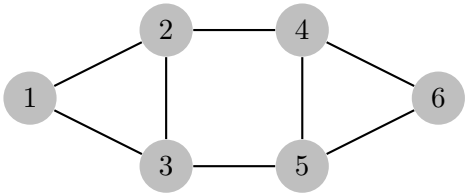


Figure 1.2. Example of undirected graph

Definition 4 (Degrees)**.** *The indegree of a node is the number of incoming arcs of that node and its outdegree is the number of its outgoing arcs. The degree of a node is the sum of its indegree and outdegree.*

Definition 5 (Adjacency List)**.** *The arc adjacency list $A(i)$ of a node i is the set of arcs emanating from that node, that is, $A(i) = \{(i,j) \in A : j \in N\}$. The node adjacency list $N(i)$ is the set of nodes adjacent to that node; in the case of a directed graph, $N(i) = \{j \in N : (i,j) \in A\}$.*

Definition 6 (Subgraph)**.** *A graph $G^{'} = (N^{'}, A^{'})$ is a subgraph of $G = (N, A)$ if $N^{'} \subseteq N$ and $A^{'} \subseteq A$. Moreover a graph $G^{'} = (N^{'}, A^{'})$ is a spanning subgraph of $G = (N, A)$ if $N^{'} = N$ and $A^{'} \subseteq A$.*

Definition 7 (Walk)**.** *A walk in a directed graph $G = (N, A)$ is a subgraph of G consisting of a sequence of nodes and arcs $i_1 - a_1 - i_2 - a_2 - ... - i_{r-1} - a_{r-1} - i_r$ satisfying the property that for all $1 \leq k \leq r-1$, either $a_k = (i_k, i_{k+1}) \in A$ or $a_k = (i_{k+1}, i_k) \in A$.*

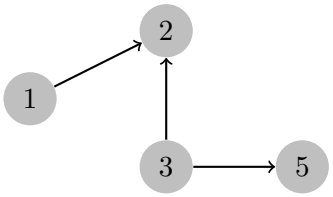


Figure 1.3. Example of walk

Definition 8 (Path)**.** *A path is a walk without any repetition of nodes.*

Definition 9 (Directed Path)**.** *A directed path is a directed walk without any repetition of nodes.*

Definition 10 (Cycle)**.** *A cycle is a path $i_1 - i_2 - ... - i_r$ together with the arc (i_r, i_1) or (i_1, i_r).*

Definition 11 (Directed Cycle)**.** *A directed cycle is a directed path $i_1 - i_2 - ... - i_r$ together with the arc (i_r, i_1).*

Definition 12 (Cycle Space)**.** *The cycle space of a graph G, denoted by $\mathcal{C}$, is the smallest set (of undirected arcs or edges) containing the $\emptyset$, all cycles in G (where each cycle is treated as a set of edges) and all unions of disjoint cycles in G.*

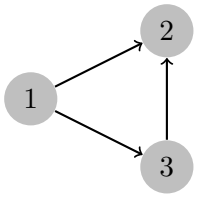


Figure 1.4. Example of cycle

Definition 13 (Acyclic Graph)**.** *A graph is acyclic if it contains no directed cycle.*

Definition 14 (Connectivity)**.** *Two nodes i and j are connected if the graph contains at least one path from node i to node j. A graph is connected if every pair of its nodes is connected; otherwise, the graph is disconnected. The maximal connected subgraphs of a disconnected network are its components.*

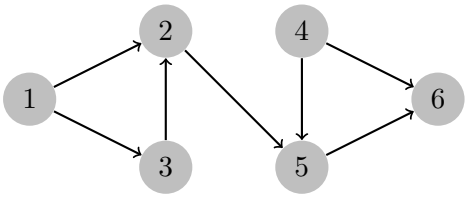


Figure 1.5. Example of connected graph

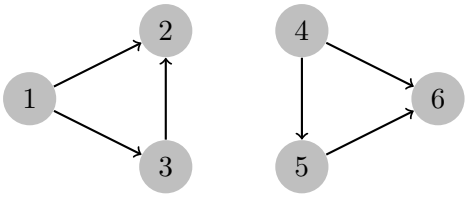


Figure 1.6. Example of disconnected graph

Definition 15 (Tree)**.** *A tree is a connected graph that contains no cycle.*

Proposition 1 (Tree properties)**.**

- *A tree on n nodes contains exactly $n-1$ arcs.*
- *A tree has at least two leaf nodes (i.e., nodes with degree 1).*
- *Every two nodes of a tree are connected by a unique path.*

Definition 16 (Spanning Tree)**.** *A tree T is a spanning tree of G if T is a spanning subgraph of G.*

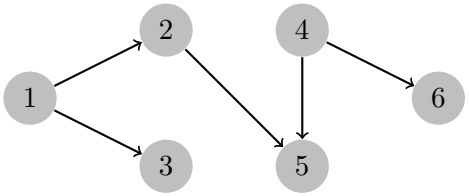


Figure 1.7. Example of spanning tree for the graph in Figure 1.5

Definition 17 (Fundamental Cycle)**.** *Let T be a spanning tree of the graph G. The addition of any non tree arc to the spanning tree T creates exactly one cycle. Any such cycle is defined a fundamental cycle of G with respect to the tree T.*

Definition 18 (Fundamental System of Cycles)**.** *Let T be a spanning tree of the graph G. The fundamental system of cycles of G with respect to T is the set of all fundamental cycles of G with respect to T.*

An important result related to the fundamental system of cycles of a graph G with respect to a spanning tree T of G, is Theorem 1. It will be useful in Chapter 5 in order to prove further results.

Theorem 1. *Let G be a connected graph and let T be a spanning tree of G. Then the fundamental system of cycles with respect to G and T forms a basis for the cycle space $\mathcal{C}$.*

Proof. Recall first that such a spanning tree T exists when G is connected. Consider any fundamental cycle C. This cycle is constructed by adding exactly one edge to T and finding the cycle that results. Thus no fundamental cycle can be expressed as the sum of any other fundamental cycles because they will all be missing (at

least) one edge. As a result, the fundamental system of cycles must be linearly independent.
Choose any element C of $\mathcal{C}$ and let $\{e_1, ..., e_r\}$ be the edges of C that do not appear in T. Further define the fundamental cycle C_i to be the one that arises as a result of adding e_i to T. The quantity $C + C_1 + ... + C_r = \emptyset$ if and only if $C = C_1 + ... + C_r$ because there are no edges in C that are not in $C_1 + ... + C_r$ and similarly no edges in $C_1 + ... + C_r$ that are not in C. It is easy to see that no edge in the set $\{e_1, ..., e_r\}$ appears in $C + C_1 + ... + C_r$ because each edge appears once in C and once in one of the C_i's and therefore, it will not appear in $C + C_1 + ... + C_r$. But this means that every edge in $C + C_1 + ... + C_r$ is an edge of T. More specifically, the subgraph induced by the edges in $C + C_1 + ... + C_r$ is a subgraph of T and thus necessarily acyclic. Every element of C induces a subgraph of G that has at least one cycle except for $\emptyset$. Thus, $C + C_1 + ... + C_r = \emptyset$. Since our choice of C was arbitrary we can see that for any C we have: $C = \alpha_1 C_1 + ... + \alpha_k C_k$ where $\{C_1, ..., C_k\}$ is the fundamental set of cycles of G with respect to T and $\forall i = 1, ..., k$:

$$\alpha_i \begin{cases} = & 1 \text{ if the non-tree edge of } C_i \text{ is found in } C \\ & 0 \text{ otherwise} \end{cases}$$

Thus, the fundamental set of cycles is linearly independent and spans $\mathcal{C}$ and so it is a basis. This completes the proof. □

Definition 19 (T-exchange/Cyclic-interchange)**.** *Let T be a spanning tree of the graph G. A T-exchange is a pair of arcs e, f where $e \in T$, $f \notin T$, and $T - e \cup f$ is a spanning tree. The weight of exchange e,f is $w(f) - w(e)$, where w is the vector of the graph weights associated with the arcs.*

We recall a combinatorial result related to spanning trees and cyclic-interchanges that will be useful in Chapter 6.

Theorem 2. *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 the tree T by at most $D(T, T^{'})$ cyclic interchanges, where:*

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

1.2 Network flow problems and main applications

In this section the main network flow problems are reported together with their most common applications. To this end the concept of flow network and flow on a network are introduced.

Definition 20 (Flow Network)**.** *A flow network is a directed graph $G = (N, A)$ with two distinguished vertices s and t called the source and the sink, respectively.*

Moreover, to each arc $(i, j) \in A$ is assigned a certain upper bound $u(i, j) \geq 0$ and lower bound $l(i, j) \geq 0$.

Definition 21 (Flow)**.** *A flow for a network $G = (N, A)$ is a function $f : V \times V \to R$, which assigns a real number to each arc (i, j). A flow f is called a feasible flow if it satisfies the following conditions:*

(i) $l(i, j) \leq f(i, j) \leq u(i, j)$, $\forall (i, j) \in A$.

These are the capacity constraints (if a capacity is ∞, then there is no upper bound on the flow value on that arc).

(ii) For all $i \in N - \{s, t\}$, the total flow into i is the same as the total flow out of i:

$$\sum_{j:(j,i)\in A} f(j, i) = \sum_{j:(i,j)\in A} f(i, j)$$

These are called the flow conservation constraints.

Minimum Cost Flow Problem

The minimum cost flow model is the most fundamental among the network flow problems. This model has several applications: the distribution of a product from manufacturing plants to warehouses, or from warehouses to retailers; the flow of raw material and intermediate goods through the various machining stations in a production line; the routing of automobiles through an urban street network; and the routing of calls through the telephone system.

Mathematical formulation

Let $G = (N, A)$ be a directed network defined by a set N of n nodes and a set A of m directed arcs. Each arc $(i, j) \in A$ has an associated cost c_{ij} that denotes the cost per unit flow on that arc. It is assumed that the flow cost varies linearly with the amount of flow. It is also associated with each arc $(i, j) \in A$ a capacity u_{ij} that denotes the maximum amount that can flow on the arc and a lower bound l_{ij} that denotes the minimum amount that must flow on the arc. Moreover at each node $i \in N$ is associated an integer number $b(i)$ representing its supply/demand. If $b(i) > 0$, node i is a supply (or excess) node; if $b(i) < 0$, node i is a demand (or deficit) node with a demand of $-b(i)$; and if $b(i) = 0$, node i is a transshipment node. The decision variables in the minimum cost flow problem are arc flows and are represented by $x_{ij} \geq 0$ $\forall (i, j) \in A$. The minimum cost flow problem is an optimization model formulated as follows:

$$\min \sum_{(i,j)\in A} c_{ij} x_{ij} \tag{1.1}$$

subject to

$$\sum_{j:(i,j)\in A} x_{ij} - \sum_{j:(j,i)\in A} x_{ji} = b(i) \quad \forall i \in N \tag{1.2}$$

$$l_{ij} \leq x_{ij} \leq u_{ij} \quad \forall (i,j) \in A \tag{1.3}$$

where $\sum_{i=1}^{n} b(i) = 0$.
We will refer to the constraint (1.2) as flow conservations constraint. For any node the first term in the above constraint represents the total outflow of the node (i.e., the flow emanating from the node) and the second term represents the total inflow of the node (i.e., the flow entering the node). The flow conservation constraint guarantees that the difference between outflow and inflow is equal the supply/demand of the node. If the node is a supply node, its outflow exceeds its inflow; if the node is a demand node, its inflow exceeds its outflow; and if the node is a transshipment node, its outflow equals its inflow. The flow must also satisfy the lower bound and capacity constraints (1.3). The flow bounds typically model physical capacities or restrictions imposed on the flows operating ranges. In most applications, the lower bounds on arc flows are zero.

Definition 22 (Augmenting cycle)**.** *A cycle W (not necessarily directed) in G is called an augmenting cycle with respect to the flow x if:*

i) by augmenting a positive amount of flow around the cycle, the flow remains feasible.
ii) the augmentation increases the flow on forward arcs and decreases the flow on backward arcs in the cycle.

Definition 23 (Incremental/Residual graph)**.** *Given a feasible flow x for the min cost flow problem on the network $G = (N, A)$, the incremental graph is a directed graph $G_x = (N, A_x)$ with arc costs c^x and arcs distinguished as follows:*

$$a^+ \in A_x^+ \quad \forall a \in A : x_a < u_a$$

$$a^- \in A_x^- \quad \forall a \in A : x_a > l_a$$

Let $A_x = A_x^+ \cup A_x^-$, the arcs in A_x have the following relationship with the arcs in A: for each $a^+ \in A_x$ $t(a^+) = t(a)$, $h(a^+) = h(a)$ and $c_a^x = c_a$; for each $a^- \in A_x$ $t(a^-) = h(a)$, $h(a^-) = t(a)$ and $c_{a^-} = -c_a$.

Observation 1. *W is an augmenting cycle with respect to a feasible flow x if and only if W corresponds to a directed cycle in the residual network G_x.*

Moreover we recall some well-known results that will be useful in Chapter 5 for proving further results.

Theorem 3 (Flow Decomposition Theorem)**.** *Every path and cycle flow has a unique representation as a non-negative flow. Conversely, every non-negative flow x can be*

represented as a path and cycle flow with the following two properties:

i) Every directed path with positive flow connects a deficit node with an excess node.

ii) At most $n + m$ paths and cycles have non-zero flow and at most m cycles have non-zero flow. If the flow x is integral, then so are the path and cycle flows which it decomposes.

Proof. The following algorithm finds the set $\mathcal{S}$ consisting of at most $n + m$ paths and cycles with non-zero flow.

$\mathcal{S} = \emptyset$

while there exists a deficit node i

Follow arcs with positive flows starting from i until either an excess node j is found or a node is visited twice.

If an excess node j is found

P is the path from i to j
The flow on the path P is $flow(P) = \min\{-b(i), b(j),$ flow of some arc in $P\}$
$\mathcal{S} = \mathcal{S} \cup P$

If a node k is visited twice

W is the cycle from k to k
The flow on the cycle W is $flow(W) = \min\{$flow of some arc in $W\}$
$\mathcal{S} = \mathcal{S} \cup W$

end while

while there exists an arc a with positive flow

Follow backward arcs with positive flows starting from a until a node is visited twice. Let W be the cycle obtained, then $flow(W) = \min\{$flow of some arc in $W\}$ and $\mathcal{S} = \mathcal{S} \cup W$

end while

end procedure

A deficit node exists if and only if an excess node exists, then after the execution of the first while block there are no more imbalanced nodes.
At each iteration either one node becomes balanced or the flow of some arc becomes 0, then there are at most $n + m$ iterations and, as a consequence, $n + m$ paths or

cycles.
Whenever a cycle is obtained the flow of some arc becomes 0, then there are at most m cycles. □

Consequence of Theorem 3 is Corollary 1.

Corollary 1. *If there are no deficit or excess nodes in the network, then a non-negative flow x can be represented as at most m cycles with non-zero flow.*

Another fundamental result that is consequence of Theorem 3 is the so called **Augmenting Cycle Theorem**.

Theorem 4 (Augmenting Cycle Theorem)**.** *Let x and x^0 be two feasible flows. Then x equals x^0 plus the flow on at most m directed cycles in $G(x^0)$. Furthermore the cost of x equals the cost of x^0 plus the cost of the flows on these augmenting cycles.*

Proof. For the hypothesis we have:

$$x = x^0 + x^{'}$$

where $x^{'}$ is a flow in the residual network $G(x^0)$. Since the residual network $G(x^0)$ does not have deficit and excess nodes, it follows, from Corollary 1, that $x^{'}$ can be decomposed in at most m directed flow cycles in $G(x^0)$. □

Shortest path problem
The shortest path problem is one of the simplest network flow problems. In this problem the goal is to find a directed path of minimum cost (or length) from a specified source node s to another specified sink node t, assuming that each arc $(i, j) \in A$ has an associated cost (or length) c_{ij}. Some of the simplest applications of the shortest path problem are to determine a path between two specified nodes of a network that has minimum length, or a path that takes least time to traverse, or a path that has the maximum reliability. It can be mathematically formulated as a minimum cost flow problem defining $b(s) = 1$, $b(t) = -1$, and $b(i) = 0$ for all other nodes. This basic model has applications in many different fields, such as equipment replacement, project scheduling, cash flow management, message routing in communication systems, and traffic flow through congested cities.

Maximum flow problem
The maximum flow problem can be seen like a complementary model to the shortest path problem. The shortest path problem models situations in which flow implies a cost but is not restricted by any capacities; in the maximum flow problem flow doesn't imply costs but is restricted by flow bounds. The maximum flow problem consists in finding a feasible solution that sends the maximum amount of flow from a specified source node s to another specified sink node t. Examples of the maximum flow problem include determining the maximum flow of petroleum products in a pipeline network, cars in a road network, messages in a telecommunication network, and electricity in an electrical network. This problem can be formulated as a minimum cost flow problem in the following way. Let $b(i)$ be equal to 0 for all $i \in N$, $c_{ij} = 0$ for all $(i, j) \in A$, and let (t, s) be an additional arc with cost $c_{ts} = -1$ and

flow bound $u_{ts} = \infty$. Then the minimum cost flow solution maximizes the flow on arc (t, s); but since any flow on arc (t, s) must travel from node s to node t through the arcs in A, the solution to the minimum cost flow problem will maximize the flow from node s to node t in the original network.

Assignment problem
The data of the assignment problem are represented by two equally sized sets N_1 and N_2, a collection of pairs $A \in N_1 \times N_2$ representing possible assignments, and a cost c_{ij} associated with each element $(i, j) \in A$. In the assignment problem the goal is to pair, at minimum possible cost, each object in N_1 with exactly one object in N_2. Examples of the assignment problem include assigning people to projects, jobs to machines, tenants to apartments, and medical school graduates to available internships. Also the assignment problem can be written as a minimum cost flow problem considering the network $G = (N_1 \cup N_2, A)$ with $b(i) = 1\ \forall i \in N_1$, $b(i) = -1\ \forall i \in N_2$, and $u_{ij} = 1\ \forall (i, j) \in A$.

Transportation problem
The transportation problem is a special case of the minimum cost flow problem with the property that the node set N is partitioned into two subsets N_1 and N_2 (of possibly unequal cardinality) so that: (1) each node in N_1 is a supply node, (2) each node N_2 is a demand node, and (3) for each arc $(i, j) \in A$, $i \in N_1$ and $j \in N_2$. The classical example of this problem is the distribution of goods from warehouses to customers. In this context the nodes in N_1 represent the warehouses, the nodes in N_2 represent customers (or, more typically, customer zones), and an arc $(i, j) \in A$ represents a distribution channel from warehouse i to customer j.

Circulation problem
The circulation problem is a minimum cost flow problem with only transshipment nodes; that is, $b(i) = 0\ \forall i \in N$. A feasible flow x must satisfy the lower and upper bounds l_{ij} and u_{ij} imposed on each arc flow variable x_{ij}. Since no exogenous flow is ever introduced to or extracted from the network, all the flow circulates around the network. The goal is to find the circulation that has the minimum cost. The design of a routing schedule of a commercial airline provides one example of a circulation problem. In this setting, any airplane circulates among the airports of various cities; the lower bound l_{ij} imposed on an arc (i, j) is 1 if the airline needs to provide service between cities i and j, and so must dispatch an airplane on this arc.

Multicommodity flow problems
The minimum cost flow problem models the flow of a single commodity over a network. Multicommodity flow problems arise when several commodities use the same underlying network. The commodities may either be differentiated by their physical characteristics or simply by their origin-destination pairs. Different commodities can have different origins and destinations, and commodities have separate flow conservation constraints at each node. However, the sharing of the common arc capacities binds the different commodities together. In fact, the essential issue addressed by the multicommodity flow problem is the allocation of the capacity of each arc to the individual commodities in a way that minimizes overall flow costs.

Multicommodity flow problems arise in many practical situations, including the transportation of passengers from different origins to different destinations within a city; the routing of nonhomogeneous tankers (in terms of speed, carrying capability, and operating costs); the worldwide shipment of different varieties of grains, from countries that produce grains to those that consume them; and the transmission of messages in a communication network between different origin-destination pairs.

Minimum spanning tree problem

A spanning tree is a tree (i.e., a connected acyclic graph) that covers all the nodes of an undirected network $G = (V, E)$ with $\mid V \mid = n$. The cost of a spanning tree is the sum of the costs (or lengths) of its edges. In the minimum spanning tree problem, the goal to identify a spanning tree of minimum cost (or length). The applications of the minimum spanning tree problem are quite diversified and include constructing highways or railroads spanning several cities; laying pipelines connecting offshore drilling sites, refineries, and consumer markets; designing local access networks; and making electric wire connections on a control panel. In its original mathematical formulation it is not written as a min cost flow problem, but as an integer linear programming problem like below.

Mathematical formulation (exponential number of constraints)

$$\min \sum_{(i,j)\in E} c_{ij}x_{ij} \tag{1.4}$$

subject to

$$\sum_{(i,j)\in E} x_{ij} = n - 1 \tag{1.5}$$

$$\sum_{(i,j)\in E(S)} x_{ij} \leq \mid S \mid -1 \ \forall S \subseteq V, \ S \neq \emptyset \tag{1.6}$$

$$x_{ij} \in \{0, 1\} \tag{1.7}$$

This formulation is the integer linear program for the problem of minimizing the total cost (or length) of a spanning tree for the network under consideration. Constraint (1.5) guarantees that the total number of edges in the spanning tree is equal to $n - 1$ as stated in Proposition 1. Constraints (1.6) are called subtour elimination constraints and ensure that no subgraph induced by S, a generic subset of the network nodes, contains a cycle. Because of these constraints, this formulation is an exponential-sized integer linear program. A possible alternative formulation for the minimum spanning tree problem, with a polynomial number of constraints, can be obtained representing it as a minimum cost flow problem with additional binary variables for each edge, representing if the edge is utilized or not, and an additional constraint which limits the total number of utilized edges to be exactly $n - 1$ (see [64]). More precisely, a network flow problem is formulated based on the same network in which one node is assumed to be a source sending $n - 1$ units of flow to all other nodes, each one requiring one unit of flow. It can be easily seen that the solution of this problem is a spanning tree.

Mathematical formulation (polynomial number of constraints)
Considering node 1 as source, the following single commodity flow problem is defined:

$$\min \sum_{e \in E} c_e x_e \tag{1.8}$$

subject to

$$\sum_{j:(1,j)\in E} f_{1j} - \sum_{j:(j,1)\in E} f_{j1} = n - 1 \tag{1.9}$$

$$\sum_{j:(i,j)\in E} f_{ji} - \sum_{j:(j,i)\in E} f_{ij} = 1, \ \forall i \in V, \ i \neq 1 \tag{1.10}$$

$$f_{ij} \leq (n-1)x_e \ \forall e = (i,j) \in E \tag{1.11}$$

$$f_{ji} \leq (n-1)x_e \ \forall e = (i,j) \in E \tag{1.12}$$

$$\sum_{e \in E} x_e = n - 1 \tag{1.13}$$

$$f_{ij}, f_{ji} \geq 0 \ \forall e = (i,j) \in E \tag{1.14}$$

$$x_e \in \{0,1\} \ \forall e \in E \tag{1.15}$$

The formulation (6.1) to (6.8) models the minimum spanning tree problem by means of a single commodity flow problem, taking one node as source of $n-1$ units of flow, while the other nodes are sinks demanding one unit of flow. Two sets of variables are introduced, the flow variables f_{ij} denoting the flow on edge (i,j) (note that although the arcs are undirected, the flow variables are directed), and the binary variables x_e denoting if the edge e is chosen in solution. Constraints (1.9) and (1.10) model the flow balances at the nodes. Constraints (1.11) and (1.12) force the flow variables associated to edges not in solution to be equal to 0. Constraint (1.13) guarantees that the topology of the solution is a spanning tree. Then, solving this formulation, one can find the minimum spanning tree of the network.

1.3 Network simplex method

The simplex method is widely used for general linear programming programs and has specific implementations for network flow problems. In practice the general simplex method, when implemented without exploiting the underlying network structure, is not a competitive solution procedure for solving the minimum cost flow problem. Fortunately, as these problems have a special structure, it was possible to adapt and interpret the core concepts of the simplex method appropriately as network operations, producing a very efficient algorithm, that is the network simplex algorithm. As explained in [3], when the original network has positive lower bound capacities, it can be transformed into a network with zero lower bound capacities. Therefore, we assume $l_{ij} = 0$ in the following.
The central concept underlying the network simplex algorithm is the notion of spanning tree solutions, that is solutions obtained by fixing the flow of every arc not in the spanning tree either at zero or at the upper bound of the arc and then

by solving only for the flow on all the arcs in the spanning tree. The procedure consists in moving from one such solution to another, at each step, introducing one new non-tree arc into the spanning tree in place of one tree arc. The name network simplex algorithm derives from the fact that the spanning trees correspond to the so-called basic feasible solutions of linear programming, and the movement from one spanning tree solution to another corresponds to the so-called pivot operation of the general simplex method.

1.3.1 Fundamentals results

As previously mentioned, the network simplex method moves among spanning tree solutions. This is based on the central result of Theorem 5.

Theorem 5 (Spanning Tree Property)**.** *The minimum cost flow problem always has an optimal spanning tree solution.*

A spanning tree solution is characterized by a partition of the arc set A in three subsets: T, the arcs in the spanning tree; L, the nontree arcs with flow equal to the lower bound; and U, the nontree arcs with flow equal to the upper bound. The triple (T, L, U) is called spanning tree structure. At each spanning tree structure it is possible to associate a spanning tree solution, setting $x_{ij} = 0 \; \forall (i,j) \in L$, $x_{ij} = u_{ij} \; \forall (i,j) \in U$ and then solving the flow conservation constraints to determine the flow values for the arcs in T. A spanning tree structure is feasible if its associated spanning tree solution satisfies all of the arcs flow bounds. A spanning tree structure is optimal if its associated spanning tree solution is an optimal solution of the minimum cost flow problem. Theorem 6 states a sufficient condition for a spanning tree structure to be optimal.

Theorem 6 (Minimum Cost Flow Optimality Conditions)**.** *A spanning tree structure (T, L, U) is optimal for the minimum cost flow problem if it is feasible and for some choice of node potentials π, the arc reduced costs c_{ij}^{π} satisfy the following conditions:*

- $c_{ij}^{\pi} = 0 \;\; \forall (i,j) \in T$
- $c_{ij}^{\pi} \geq 0 \;\; \forall (i,j) \in L$
- $c_{ij}^{\pi} \leq 0 \;\; \forall (i,j) \in U$

The reduced cost $c_{ij}^{\pi} = c_{ij} - \pi(i) + \pi(j)$ for a nontree arc $(i,j) \in L$ denotes the change in the cost of the flow that can be realized by sending 1 unit of flow from node i to node j through the arc (i,j) appropriately modifying the flow on the arcs in the tree path between node i and node j. The network simplex algorithm maintains a feasible spanning tree structure and moves from one spanning tree structure to

another until it finds an optimal structure. At each iteration, the algorithm adds one arc to the spanning tree in place of one of its current arcs. The entering arc is a nontree arc violating its optimality condition. Because of its relationship to the primal simplex algorithm for the linear programming problem, this operation of moving from one spanning tree structure to another is known as a pivot operation, and the two spanning trees structures obtained in consecutive iterations are called adjacent spanning tree structures.

1.3.2 Algorithm scheme

The network simplex algorithm maintains a feasible spanning tree structure at each iteration and successively transforms it into an improved spanning tree structure until it becomes optimal. The essential steps of the method are described in the following scheme:

1. Determine an initial feasible tree structure (T, L, U);
Let x be the flow and π be the node potentials associated with this tree structure;
2. **while**(some nontree arc violates the optimality conditions)
3. select an entering arc (k, l) violating its optimality condition;
4. add arc (k, l) to the tree and determine the leaving arc (p, q);
5. perform a tree update and update the solutions x and π;
6. **end while**

Each step of this procedure can be implemented in different ways. Moreover, according with the specific network flow problem under consideration, the network simplex method can be specialized in several versions (see [3]).

Chapter 2

Emerging optimization problems in Telecommunication networks

In this chapter new emerging optimization problems on Telecommunication (TLC) networks are described, mathematically modeled, and solved on a real or realistic scenarios. These results published in [21], [23], [22] and [5] are significative from the point of view of the application, but it will be pointed out that the multiple objective nature of these problems calls for new advances also from the methodological viewpoint for fundamental problems in multi-objective optimization.

2.1 Managing the Energy-Lifetime Trade-off in Backbone Networks

This section contains the main results from an operations research perspective of the paper [22] published in Transactions on Networking, 2017. During the last few years, the problem of "green" backbone networks has gained significant importance, starting from the seminal work of Gupta and Sigh [44]. The goal of green networking is to exploit the power management policies to reduce the network energy cost. In particular, a Sleep Mode (SM) is defined: when a SM state is set for a device, the other devices that remain powered on have to sustain the traffic between source and destination nodes. In this context, different works (see for example [1], [42], [24], [17]) have investigated the management of Internet Provider (IP) backbone networks by adopting SM. The main outcome of these works is that networks with SM capabilities are able to save energy, due to the fact that the traffic varies between day and night, resulting in a large number of resources that can be put in SM during the off-peak hours.
However, the impact of SM on the reliability of network devices is an open issue [82], [20]. In particular, there are two opposite effects influencing the lifetime of network devices [20]: the SM duration, which tends to increase the lifetime, and the change in the power state (from SM to full power and viceversa), which instead decreases the lifetime. In general, when a network device experiences a failure, the traffic

flowing on it may be dropped, resulting in a Quality of Service (QoS) degradation for users. Additionally, reparation costs are incurred, which may involve even the replacement of the whole device. In particular, the reparation costs may even exceed the monetary savings derived from SM [81]. All these facts suggest that the device lifetime plays a crucial role in determining the efficiency of SM, and the energy saving may not be the only metric to prove the effectiveness of a SM-based approach. In [19] a simple model to evaluate the lifetime increase/decrease of network devices taking into account the SM duration and its frequency, has been proposed. The model shows that the energy-aware algorithms have an impact on the device lifetime, which may be positive or negative, depending on the Hardware (HW) components used to build the device and on the strategy adopted for choosing which devices should be put in SM. In this context, a natural and interesting problem is related to the possibility of managing SM while always limiting the impact on the lifetime. In this section, to this end, a mathematical formulation of the lifetime-aware problem, over a backbone network, will be described and effectively solved.

2.1.1 Related work

Previous works on green networking are mainly focused on reducing the energy-consumption of devices (see e.g. [13] and [28] for detailed surveys). To this end, different approaches have been proposed, ranging from centralized solutions [42], [24], [17] (i.e., one central controller that decides which elements in the network to put in SM) to distributed solutions [25], [52], [83] (i.e., each node decides which of its own devices to put in SM). All these solutions prove the effectiveness of saving energy by exploiting two main features: i) the fact that current backbone networks are normally over-dimensioned, and ii) the variation of traffic between peak and off-peak hours. However, the impact on the device lifetime is not considered. Traditionally, the node lifetime has been investigated in the context of Wireless Sensors Networks (WSNs) [15], [57], where each node of a WSN is powered by a battery, and therefore it is important to prolong the lifetime of the network, i.e., to avoid the case in which some devices exhaust the battery and then source and destination nodes become disconnected. In general, there are different constraints that may limit the application of power management policies in a network. Usually, guaranteeing protection is one of them [4], [38], [63]. Other constraints include the limitation in the increase of the path length [2], [84], the network delay [61], or the signal quality [16].
In this context, two main reasons suggest to include the lifetime: i) a lifetime decrease may increase the reparation and replacement costs in order to fix the failed devices, and ii) when a device fails, the QoS may be heavily impacted. According with this observation a methodology in order to manage the lifetime while allowing devices to exploit power management primitives is proposed. Such a methodology is based on the idea that the lifetime-awareness should be integrated in the process of deciding when and for how long a power state should be applied to a device.

2.1.2 Modeling the device lifetime

Focusing on the links of an IP backbone network, the generic failure rate for a link at full power is denoted with γ^{on}. When SM is applied to the link, the new failure rate γ^{tot} is defined as:

$$\gamma^{tot} = \gamma^{on}(1 - \frac{\tau^s}{T}) + \gamma^s \frac{\tau^s}{T} + \frac{f^{tr}}{N^F}$$

where τ^s is the total time in SM during time period T, γ^s is the failure rate in SM (which is supposed to be lower than γ^{on}), f^{tr} is the power switching rate between full power and SM, and N^F is a parameter called number of cycles to failures. The main assumptions behind such a definition are that failures are statistically independent from each other, and their effects are additive. In order to evaluate the lifetime increase/decrease w.r.t. the always on solution (i.e., all links powered on), we define a metric called acceleration factor (AF) (see [20]), which is the ratio between the failure rate with SM γ^{tot} and the one at full power γ^{on}. The AF metric is lower than 1 if the link lifetime is increased compared to the always on solution. On the contrary, a value larger than 1 means that the lifetime is decreased compared to the always on case. More formally the acceleration factor is defined as follows:

$$AF = \frac{\gamma^{tot}}{\gamma^{on}} = 1 - (1 - AF^s)\frac{\tau^s}{T} + \chi f^{tr}$$

where AF^s is defined as $\frac{\gamma^s}{\gamma^{on}}$, which is always lower than 1 since the failure rate in SM γ^s (by neglecting the negative effect due to power state transitions) is always lower than the failure rate at full power γ^{on}. Moreover, χ is defined as $\frac{1}{\gamma^{on} N^F}$, which acts as a weight for the power switching rate f^{tr}. The AF is then composed of two terms: the first one which tends to increase the lifetime (i.e., the longer period of SM tends to increase this term which is negative), while the second one instead tends to decrease the lifetime (i.e.,the more often power state transitions occur, the higher this term is). Moreover, the model is composed by parameters AF^s and χ, which depend solely on the HW components used to build the link, while parameters τ^s and f^{tr} depend instead on the realization of SM.

2.1.3 Mathematical Formulation

The goal of the problem under consideration is to minimize the AF in a network exploiting link SM. That is, given the set of nodes and links in the network, and the traffic for each time slot, the objective is the minimization of the average AF over the whole time period, subject to connectivity and maximum link utilization for each time slot. More formally, let $G = (V, E)$ be the network topology. Let V be the set of the network nodes, while E is the set of network links, being $\mid V \mid = N$ and $\mid E \mid = L$, respectively. Let us denote with T the total amount of time under consideration. T is divided in K time slots of period δ_T. Moreover, let $t^{s,d}(k) \geq 0$ be the traffic demand from node s to node d during slot k. The problem described above is mathematically modeled as follows:

$$\min \frac{1}{L} \sum_{(i,j) \in E} AF_{i,j} \tag{2.1}$$

The goal is to minimize the average AF of the links in the network. Moreover the traffic has to be routed in the network for each time slot through the following demand satisfaction and flow conservation constraints:

$$\sum_{j:(i,j)\in E} f_{i,j}^{s,d}(k) - \sum_{j:(j,i)\in E} f_{j,i}^{s,d}(k) = \begin{cases} t^{s,d}(k) \text{ if } i = s \\ -t^{s,d}(k) \text{ if } i = d \\ 0 \text{ if } i \neq s,d \end{cases} \quad \forall i,s,d,k \tag{2.2}$$

where $f_{i,j}^{s,d}(k) \geq 0$ is the amount of flow from s to d that is routed through link (i,j) during slot k. The total amount of flow $f_{i,j}(k) \geq 0$ on each link for each slot is given by:

$$f_{i,j}(k) = \sum_{s,d} f_{i,j}^{s,d}(k) \quad \forall (i,j) \in E, \ \forall k \tag{2.3}$$

Let $c_{i,j} > 0$ be the capacity of the link (i,j) and $\alpha \in (0,1]$ be the maximum link utilization that can be tolerated, respectively. Moreover, the binary variable $x_{i,j}(k)$, which takes value of 1 if link (i,j) is powered on during slot k, zero otherwise, is introduced. The total amount of flow has to be smaller than $c_{i,j}$, scaled by the maximum link utilization α. Such a condition is imposed by the following capacity constraints:

$$f_{i,j}(k) \leq \alpha c_{i,j} x_{i,j}(k) \quad \forall (i,j) \in E, \ \forall k \tag{2.4}$$

Note that these constraints impose also the fact that a link has to be powered on if the flow on it is larger than zero. The link state has to be the same in both directions (it is assumed that SM can be set only if the links in both directions are not carrying any traffic):

$$x_{i,j}(k) = x_{j,i}(k) \quad \forall (i,j) \in E, \ \forall k \tag{2.5}$$

Then the binary variable $z_{i,j}(k)$, which takes value of 1 if link (i,j) has experienced a power state transition from slot $(k-1)$ to slot k, zero otherwise, is defined. The value of $z_{i,j}(k)$ is set with the following two constraints:

$$\begin{cases} x_{i,j}(k) - x_{i,j}(k-1) \leq z_{i,j}(k) \\ x_{i,j}(k-1) - x_{i,j}(k) \leq z_{i,j}(k) \end{cases} \quad \forall (i,j) \in E, \ \forall k \tag{2.6}$$

Then the integer variable $C_{i,j} \geq 0$, which counts the total number of transitions for each link during the whole time period is introduced:

$$C_{i,j} = \sum_{k=1}^{K} z_{i,j}(k) \quad \forall (i,j) \in E \tag{2.7}$$

Additionally, the variable $\tau_{i,j}^{s} \geq 0$, which instead computes the total time in SM for each link, is defined:

$$\tau_{i,j}^{s} = \sum_{k=1}^{K} [1 - x_{i,j}(k)] \delta_T \quad \forall (i,j) \in E \tag{2.8}$$

The variable $AF_{i,j} \geq 0$ to compute the total AF for each link is given by:

$$AF_{i,j} = [1 - (1 - AF^s_{i,j})\frac{\tau^s_{i,j}}{T} + \chi_{(i,j)}\frac{C_{i,j}}{2}] \quad \forall (i,j) \in E \tag{2.9}$$

The variable $C_{i,j}$ is divided by 2 because a power cycle is always composed by at least two transitions (i.e., from full power to SM, and then from SM to full power). The formulation above includes both integer variables and continuous ones. As a result, it belongs to the class of Mixed Integer Linear Programming (MILP) problems. For this model, a set of inequalities which depend on the specific structure of our problem (following a similar procedure to [9]) has been identified. In this way, the search space was reduced in order to speed up the solution procedure while limiting the amount of time required to generate such inequalities. A more formal description of these inequalities is given below.

Valid inequalities

Let B be the set of source nodes in the network and let $\delta^+(B)$ be the set of the outgoing edges from B. Then, focusing on the constraints (2.2), (2.3), (2.4) limited to the edges of the set $\delta^+(B)$:

$$\sum_{s:(s,j)\in E} f^{s,d}_{s,j}(k) - \sum_{j:(j,s)\in E} f^{s,d}_{j,s}(k) = t^{s,d}(k) \quad \forall s \in B,\ \forall k,\ \forall d \tag{2.10}$$

$$f_{s,j}(k) \geq \sum_d f^{s,d}_{s,j}(k) \quad \forall j : (s,j) \in E,\ \forall s \in B,\ \forall k \tag{2.11}$$

$$f_{s,j}(k) \leq \alpha c_{s,j} x_{s,j}(k) \quad \forall j : (s,j) \in E,\ \forall s \in B,\ \forall k \tag{2.12}$$

Lemma 1. *Given Eq. (2.10), (2.11), (2.12) the following valid inequalities hold:*

$$\sum_{(i,j)\in\delta^+(B)} c_{i,j} x_{i,j}(k) \geq \frac{1}{\alpha} \sum_{s\in B,d} t^{s,d}(k) \quad \forall k \tag{2.13}$$

Proof. From constraint (2.10) we can write:

$$\sum_{s:(s,j)\in E} f^{s,d}_{s,j}(k) \geq t^{s,d}(k) \quad \forall s \in B,\ \forall k,\ \forall d \tag{2.14}$$

If we consider the sum on all destinations we obtain:

$$\sum_d \sum_{s:(s,j)\in E} f^{s,d}_{s,j}(k) \geq \sum_d t^{s,d}(k) \quad \forall s \in B,\ \forall k \tag{2.15}$$

From constraints (2.11) and (2.12) it is also possible to write:

$$\sum_d \sum_{s:(s,j)\in E} f^{s,d}_{s,j}(k) = \sum_{s:(s,j)\in E} \sum_d f^{s,d}_{s,j}(k) \leq$$

$$\sum_{s:(s,j)\in E} f_{s,j}(k) \leq \alpha \sum_{j:(s,l)\in E} c_{s,j} x_{s,j}(k) \quad \forall s \in B,\ \forall k \tag{2.16}$$

Additionally, from constraints (2.15) and (2.16) we obtain:

$$\alpha \sum_{j:(s,l)\in E} c_{s,j} x_{s,j}(k) \geq \sum_{s\in B,d} t^{s,d}(k) \forall s \in B, \ \forall k \tag{2.17}$$

or equivalently:

$$\sum_{j:(s,l)\in E} c_{s,j} x_{s,j}(k) \geq \frac{1}{\alpha} \sum_{d} t^{s,d}(k) \forall s \in B, \ \forall k \tag{2.18}$$

If we consider the sum on all sources we can write:

$$\sum_{s\in B} \sum_{j:(s,l)\in E} c_{s,j} x_{s,j}(k) \geq \frac{1}{\alpha} \sum_{s\in B,d} t^{s,d}(k) \quad \forall k \tag{2.19}$$

that is:

$$\sum_{(i,j)\in \delta^+(B)} c_{i,j} x_{i,j}(k) \geq \frac{1}{\alpha} \sum_{s\in B,d} t^{s,d}(k) \quad \forall k \tag{2.20}$$

□

The formulation obtained adding the set of valid inequalities (2.13), as shown by the experimental results in the next section, is an enhanced formulation that speeds up the solution procedure.

2.1.4 Experimental Results

The reference scenario has been provided by Orange-FT (see [54]). The operator has provided the topology in terms of nodes and links, link capacities, routing weights, and the traffic variation over one working day, with a time granularity of one hour between one traffic matrix and the following one (see [53] for a detailed description of this scenario). Moreover, the maximum link utilization is set to 50% of the link capacity, as suggested by the operator.
A time period T equal to four days has been considered. Unless otherwise specified, six different traffic matrices equally spaced for each day have been used (always including the peak traffic matrix in the matrix set). Moreover, these matrices were repeated over the different days. This is a conservative assumption, since the traffic during weekends may be lower compared to working days, but the obtained results are representative. Moreover the same HW parameters for the links were assumed, i.e., all links are deployed with similar devices in terms of HW characteristics.
The original formulation has been solved with CPLEX solver (version 12.6) on a high performance computing cluster, composed of four nodes, each of them with 32 cores and 64 GB of RAM, for a total computing power of around 1.5 TeraFlops/s. The average AF, the time required to retrieve the solution, and the optimality gap were considered as performance metrics. Table 2.1 reports the obtained results (the optimality gap is expressed in percentage), for different HW parameters $AF^s_{i,j}$ and $\chi_{i,j}$. Recall that $AF^s_{i,j}$ is the AF of the device when a SM state is applied (without considering transitions), while $\chi_{i,j}$ is the weight for the frequency of power state changes. Due to the fact that measurements of $AF^s_{i,j}$ and $\chi_{i,j}$ values are not

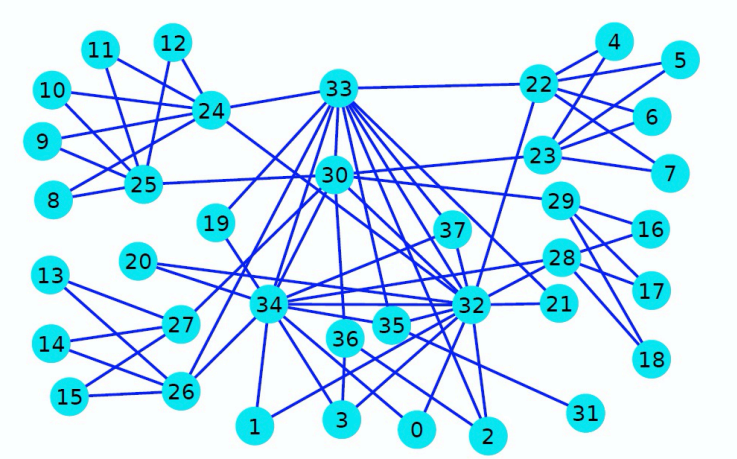


Figure 2.1. Orange-FT network topology

yet available in the literature, a sensitivity analysis w.r.t. both of them has been preformed. Moreover, a maximum time limit of 24h for retrieving the solution from the optimization solver has been imposed.

Table 2.1. Optimization Results

$AF_{(i,j)}^{sleep}$	$\chi_{i,j}$	Objective	Gap(%)	Time
0.2	0.01	0.71	4.18	24h
0.2	0.05	0.79	2.68	24h
0.2	0.1	0.87	2.4	24h
0.2	0.5	0.94	0	46’ 32”
0.2	1	0.94	0	41’ 18”
0.5	0.01	0.83	1.98	24h
0.5	0.05	0.90	2.21	24h
0.5	0.1	0.96	0.21	24h
0.5	0.5	0.97	0	44’ 32”
0.5	1	0.97	0	22’ 33”
0.8	0.01	0.94	0.65	24h
0.8	0.05	0.99	0	1h 10’
0.8	0.1	0.99	0	1h 36’
0.8	0.5	0.99	0	53’
0.8	1	0.99	0	51’ 20”

In eight out of the fifteen runs we found the optimal solution, in the other cases we have a gap less or equal to 4.18%.
In Table 2.2 the optimization results of the improved model are shown. As we can see, the convergence to the optimum is faster with respect to the original model. In the other cases the gap between the objective function and the lower bound found within the time limit, is lower.

Table 2.2. Optimization Results (with valid inequalities)

$AF^{sleep}_{(i,j)}$	$\chi_{i,j}$	Objective	Gap(%)	Time
0.2	0.01	0.71	2.60	24h
0.2	0.05	0.79	1.56	24h
0.2	0.1	0.87	1.36	24h
0.2	0.5	0.94	0	30 ' 42"
0.2	1	0.94	0	13' 18"
0.5	0.01	0.83	1.48	24h
0.5	0.05	0.90	1.18	24h
0.5	0.1	0.96	0.03	24h
0.5	0.5	0.97	0	20' 30"
0.5	1	0.97	0	9' 28"
0.8	0.01	0.94	0.53	24h
0.8	0.05	0.99	0	22' 43"
0.8	0.1	0.99	0	23' 3"
0.8	0.5	0.99	0	12' 38"
0.8	1	0.99	0	10' 6"

For low values of $AF^s_{i,j}$ and $\chi_{i,j}$, the problem is very challenging to be solved optimally: this is due to the fact that the gain in putting devices in SM is high, while at the same time the penalty for introducing transitions is low. Thus, the only constraint limiting the AF decrease is the traffic, which imposes to put at full power different links during peak hours. However, traffic varies during the day, therefore the set of links in SM is varied too. As a consequence, the solver always tries to maximize the number of links in SM, resulting in a low average AF and a high time required to obtain the solution. However, we can see that in all cases, the maximum gap is always below 2.60%. Moreover, when $\chi_{i,j}$ is increased, the penalty associated to the power state changes becomes not negligible. This has an impact on the objective function (resulting in a higher AF), but also on lower computation time, due to the fact that the set of links in SM changes less frequently with time. Additionally, when $AF^s_{i,j}$ is increased, we can clearly see that the AF tends to increase, since the gain for putting the devices in SM is lower. However, the AF is always lower than one, meaning that the lifetime has been increased compared to the solution in which all links are always powered on. For example, when $AF^s_{i,j} = 0.2$ and $\chi_{i,j} = 0.01$ [1/h], the AF is equal to 0.71, resulting in a lifetime increase of almost 30%.
These results show that both the HW and sleep mode parameters play a crucial role for influencing the lifetime. Moreover, they show that the lifetime of a network can be effectively managed applying SM policies.

2.2 Optimal Sustainable Management of Backbone Networks

This section contains the main results from an operation research perspective of the paper [5] published in IEEE, International Conference on Optical Networks, 2016.

As previously mentioned, even though the reduction of costs in terms of energy brought by sleep mode approaches has been already recognized and accepted by the research community, little efforts have been performed so far to understanding what are the implications of adopting this type of solution in an operator network. In particular, the transition between sleep mode and full power, applied regularly to the devices in a network, may dramatically reduce their lifetime [20]. When the devices decrease their lifetime, they need to be fixed more frequently, thus increasing the associated reparation/replacement costs [82]. As a result, there is a trade-off between the amount of energy that can be saved in a network and the maximum admissible lifetime degradation [21]. Additionally, another effect that has to be considered when adopting energy-efficient solutions is the impact on the Quality of Service (QoS) to users. More in depth, if users are served by few network devices, their experienced QoS may be low compared to the case in which all devices are always powered on. A user experiencing a low QoS may then decide to migrate to another operator, thus again representing a loss for its original operator. In this context, a natural problem that arises is the possibility to trade between network energy savings, device fixing costs, and user QoS in a backbone network. To this end a mathematical formulation in which the total sustainability of a backbone network is maximized, is here proposed. In order to reflect all the aspect above mentioned the total sustainability is expressed as a metric encompassing electricity costs, lifetime costs, and users QoS, as shown in the next section.

2.2.1 Problem Formulation

A backbone network composed of source/destination nodes and purely transport nodes was considered. It was also assumed that the links capacity and the traffic demand by all source/destination node pairs for each time period are given. The objective is to maximize the total sustainability of the network, by jointly considering the users utility, the device fixing costs, and the network energy costs. More in depth, the users utility is defined as a revenue for the operator to serve users at a given rate. The higher is the rate for serving users, the higher is also their utility. In this way, also the QoS for serving the users has been taken into account. By assuming that time is divided in time slots of fixed duration,the target is the maximization of the sustainability in each time slot by acting on the power state of each link in the topology. Each link can be at full power or in sleep mode (SM). More formally, let $G = (V, E)$ be the graph representing the network infrastructure. Let V be the set of the network nodes, while E the set of the network links. We assume $\mid V \mid = N$ and $\mid E \mid = L$. Let $c_{i,j} > 0$ be the capacity of the link (i, j) and $\alpha \in (0, 1]$ the maximum link utilization that can be tolerated. It is assumed that the total period of time under consideration is divided in time slots of duration δ_t. Let k be the current time slot index. The mathematical formulation is then introduced by means of different set of variables. Focusing on traffic, it is assumed a variable amount of traffic for each source s and each destination d. Let $t^{sd}_{min}(k)$ be the minimum amount of traffic for node pair $s - d$ at time slot k. Similarly, let $t^{sd}_{max}(k)$ be the maximum amount of traffic between s and d at time slot k. The continuous variables $\lambda^{sd}(k)$ denote the actual amount of traffic assigned to pair $s - d$ at k. Additionally, $f^{s,d}_{i,j}(k) \geq 0$ is the amount of flow from s to d that is routed through link (i, j) during current time

slot k. Similarly, $f_{i,j}(k) \geq 0$ is the total amount of flow on link (i,j) during slot k. Given the previous definitions, the problem is formulated as:

$$\max[U_{tot}(k) - (C_E(k) + C_R(k))] \tag{2.21}$$

The goal is to maximize the total sustainability of the system at each time slot k. The objective function is then given by the difference between the total utility and the total cost at time slot k. This latter is represented by the sum of the total energy cost and the total fixing cost at time slot k.

$$\sum_{j:(i,j)\in E} f_{i,j}^{s,d}(k) - \sum_{j:(j,i)\in E} f_{j,i}^{s,d}(k) = \begin{cases} \lambda^{s,d}(k) \text{ if } i = s \\ -\lambda^{s,d}(k) \text{ if } i = d \\ 0 \text{ if } i \neq s,d \end{cases} \quad \forall i \in V \tag{2.22}$$

The flow conservation constraints guarantee that the traffic is correctly routed in the network. Moreover the following constraints are imposed on the variable $\lambda^{sd}(k)$:

$$\lambda^{sd}(k) \leq t_{max}^{sd}(k) \quad \forall s,d \tag{2.23}$$

$$\lambda^{sd}(k) \geq t_{min}^{sd}(k) \quad \forall s,d \tag{2.24}$$

The total amount of flow on each link is then given by:

$$f_{i,j}(k) = \sum_{s,d} f_{i,j}^{s,d}(k) \quad \forall (i,j) \in E \tag{2.25}$$

The total amount of flow is limited to be smaller than the link capacity:

$$f_{i,j}(k) \leq \alpha c_{i,j} x_{i,j}(k) \quad \forall (i,j) \in E \tag{2.26}$$

where $x_{i,j}(k)$ is a binary variable which takes value one if the link (i,j) is powered on during slot k, zero otherwise.
Considering the users utility, let U_{min} and U_{max} be a minimum and a maximum utility value, respectively. Moreover, let λ_{min}^{sd} and λ_{max}^{sd} be the minimum and maximum thresholds for $\lambda^{sd}(k)$. The resulting utility variable $U_{s,d}$ for users requesting traffic from node s to node d is then computed as:

$$U_{s,d} = \begin{cases} U_{min} \text{ if } \lambda^{sd}(k) \leq \lambda_{min}^{sd} \\ U_{min} + (U_{max} - U_{min}) \log_2(1 + \frac{\lambda^{sd}(k) - \lambda_{min}^{sd}}{\lambda_{max}^{sd} - \lambda_{min}^{sd}}) \text{ if } \lambda_{min}^{sd} \leq \lambda^{sd}(k) \leq \lambda_{max}^{sd} \\ U_{max} \text{ if } \lambda^{sd}(k) \geq \lambda_{max}^{sd} \end{cases} \tag{2.27}$$

The previous expression assumes that the user utility scales logarithmically with the actual amount of served traffic $\lambda^{sd}(k)$. In this way, the increment of users utility

experienced at lower rates tends to be rewarded. The total utility variable $U_{tot}(k)$ at time slot k is then computed as:

$$U_{tot}(k) = \sum_{s,d} U_{s,d} + U_{tot}(k-1) \tag{2.28}$$

where $U_{tot}(k-1)$ is an input parameter representing the utility at previous time slot $k-1$. Focusing on the energy costs, first it is imposed the fact that, if the link (i,j) is put in SM, also the link (j,i) has to be powered off:

$$x_{i,j}(k) = x_{j,i}(k) \quad \forall (i,j) \in E \tag{2.29}$$

Then the total energy cost $C_E(k)$ at time slot k is computed as:

$$C_E(k) = c_{KWh}\delta_t \sum_{i,j} x_{i,j}(k) P_{i,j} + C_E(k-1) \tag{2.30}$$

where c_{KWh} is the hourly electricity cost, $P_{i,j}$ is the power consumption of the link (i,j) when it is powered on, and $C_E(k-1)$ is the energy cost at previous time slot $k-1$.
Eventually, focusing on the fixing costs, by adopting the same failure model of [20], let $\xi_{i,j}(k)$ be a binary variable which takes value one if the link (i,j) has experienced a power state transitions from slot $k-1$ to slot k, zero otherwise. $\xi_{i,j}(k)$ can be set by the same constraint imposed on the variables $z_{i,j}(k)$ of the model presented in the Section 2.1.3:

$$\begin{cases} x_{i,j}(k) - x_{i,j}(k-1) \leq \xi_{i,j}(k) \\ x_{i,j}(k-1) - x_{i,j}(k) \leq \xi_{i,j}(k) \end{cases} \quad \forall (i,j) \in E \tag{2.31}$$

Moreover, let $R_{i,j}(k) \geq 0$ be integer variable counting the number of power state transitions for the link (i,j) up to time slot k.

$$R_{i,j}(k) = \xi_{i,j}(k) + R_{i,j}(k-1) \quad \forall (i,j) \in E \tag{2.32}$$

where $R_{i,j}(k-1)$ is an input parameters storing the number of transitions up to previous time slot $k-1$. Additionally, the continuous variable $\tau_{i,j}(k)$, which stores the total amount of time in SM for the link (i,j) up to time slot k, are introduced:

$$\tau_{i,j}(k) = (1 - x_{i,j}(k))\delta_t + \tau_{i,j}(k-1) \quad \forall (i,j) \in E \tag{2.33}$$

where $\tau_{i,j(k-1)}$ is the total time in SM for the link (i,j) up to previous time slot $k-1$. Given the number of transitions $R_{i,j}(k)$ and the total time in SM $\tau_{i,j}(k-1)$, like in the model of Section 2.1.3, the metric AF for the link (i,j) at time slot k is expressed as:

$$AF_{i,j}(k) = [1 - (1 - AF^s_{i,j})\frac{\tau_{i,j}(k)}{T(k)} + \chi_{(i,j)}\frac{R_{i,j}(k)}{2}] \quad \forall (i,j) \in E \tag{2.34}$$

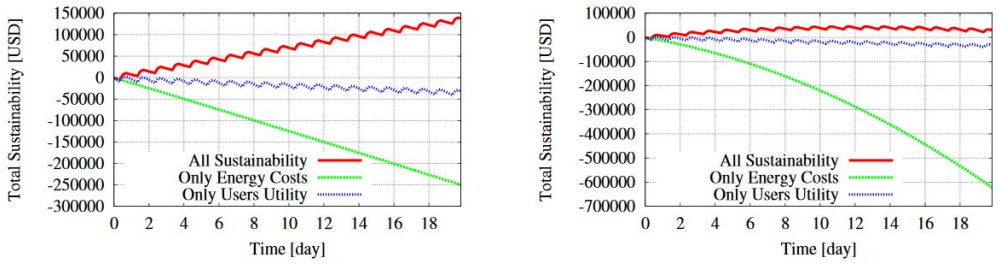
where $AF^s_{(i,j)}$ and $\chi_{(i,j)}$ are hardware input parameters, and $T(k)$ is the amount of time from the first time slot to time slot k . The total fixing costs $C_R(k)$ at time slot k are then computed as:

$$C_R(k) = C_r \delta_t \sum_{i.j} AF_{i,j}(k)\theta^{on}_{i,j} + C_R(k-1) \tag{2.35}$$

where C_r is the hourly cost for fixing a link, $\theta^{on}_{i,j}$ is the failure rate for a link always powered on, and $C_R(k-1)$ are the fixing costs up to previous time slot $k-1$.
The resulting model is a mixed integer non-linear programming model, that has been converted into a mixed integer linear programming formulation by means of a linear approximation of the logarithmic function.

2.2.2 Results

The scenario on which the model has been tested is the same described in Section 2.1.4. The network is composed of 38 nodes and 72 bidirectional links. Additionally, the traffic between each node pair and the amount of capacity installed on each link are provided in the scenario. The network is dimensioned to satisfy a maximum link utilization equal to 50% of the link capacity (during the peak hour). Moreover, the information about variation of traffic over time is also provided. More in depth, a daily traffic profile (which is repeated across a set of 20 days), and a time slot granularity of 4 hours, i.e., there are 6 time slots in a day, were considered. Focusing on power consumption, the same model of [54] has been used. Moreover, the power consumption of a link is assumed to be negligible when it is put in sleep mode. Eventually, a cost of electricity c_{KWh} equal to 0.16 [USD/kWh] has been assumed (see [20]). As regards the parameters related to the fixing costs, $AF^s_{(i,j)}$ has been set equal to 0.5, since it is assumed that the lifetime when the device is in SM is doubled compared to the case in which it is always powered on. Focusing on the other hardware parameter $\chi_{(i,j)}$, initially its value has been set equal to 0.001. Recall that this parameter acts as a weight for the number of power state transitions. With this setting, the impact of power state transition is assumed to be not so high, i.e., the device can sustain hundreds of transitions before introducing a significative lifetime decrease. Additionally, the failure rate for a link always powered on $\theta^{on}_{i,j}$ is set to 1/87600 [1/h] like in [20], which corresponds to a lifetime equal to 10 years. Eventually, the fixing cost is equal to 190 kUSD in accordance to [20]. Focusing on the user utility, λ^{sd}_{min} has been set equal to 0.0003 Gbps $\forall s, d$, $\lambda^{sd}_{max} = 2.92$ Gbps $\forall s, d$, $U_{min} = 0.033$ USD and $U_{max} = 114$ USD, respectively. With this setting, the users utility is larger than the energy costs during the peak hour (which justifies the deployment of the network), while it is lower than the energy costs during the off-peak hour and with all the links powered on (which justifies the application of energy-saving techniques).



(a) $\chi = 0.001$ (b) $\chi = 0.1$

Figure 2.2. Total Sustainability vs. Time

On this scenario the model presented has been implemented in CPLEX (version 12.6) and tested. Moreover, the following comparisons have been also considered: i) only minimization of energy costs (i.e., a classical energy-saving approach), and ii) only maximization of users utility (i.e., a solution in which all devices are always powered on). In these two latter cases, the total sustainability is then computed off-line at the end of the optimization process. Figure 2.2(a) reports the sustainability vs. time for the three formulations. Interestingly, the minimization of energy-costs tends to decrease the sustainability, i.e., at the end of the considered 20 days period the total sustainability is negative, meaning that the operator has paid a high cost rather than achieving a revenue. This is due to two main reasons: i) the lifetime degradation triggered by frequent full power/sleep mode transitions, and ii) the fact that the revenue for serving the users is not considered. Moreover, the maximization of users utility tends to have an almost constant and negative sustainability. This is due to the fact that this solution does not consider the energy costs, resulting in an electricity waste. Eventually, the proposed approach, targeting the whole sustainability (i.e., users utility, device fixing costs, and energy costs), always brings a revenue for the operator, resulting in a final sustainability of more than 150000 USD. In the following, the impact of increasing the fixing costs has been investigated, by setting a value of $\chi = 0.1$ (Figure 2.2(b)). Recall that χ is an hardware parameter acting as a weight for power state transitions in Eq. (2.34), thus potentially increasing the impact of fixing costs. In this case, the minimization of energy costs tends to notably decrease the sustainability at the end of the 20 days period. Clearly, the formulation targeting the maximization of users utility is not influenced by χ, since no transitions are introduced. Eventually, the maximization of total sustainability tends to achieve revenues also in this case. This is due to the fact that the fixing costs, the users utility, and the energy costs are jointly taken into account. According with these results, the proposed formulation outperforms both a classical approach based on the maximization of energy savings and a solution always maximizing the users utility.

2.3 Optimal Superfluid Management of 5G Networks

This section contains the main results from an operation research perspective of the paper [23] published in IEEE, International NetSoft Conference, 2017. The Internet

is becoming an ever increasing pervasive technology. According to different studies, new services like High Definition (HD) videos, tactile applications (see [36]), Internet of Things (IoT) (see [10]), and extremely low delay applications will dominate the scene in the forthcoming years. In addition to this, the number of users will continue to notably increase, especially from growing economies. As a result, the network itself will have to evolve from a monolithic architecture towards a converged, flexible, and high performance solution. To this end, new paradigms, like Network Function Virtualization (NFV) [56], have been proposed in the last years. Moreover, several initiatives are currently devoted to the design of 5G networks (see [8]), which are expected to turn into reality by 2020. In this context a so called superfluid approach has been defined, meaning that network functions and services are decomposed into reusable components, denoted as Reusable Functional Blocks (RFBs), which are deployed on top of physical nodes. RFBs have notable features, including: i) RFBs chaining, in order to implement more complex functionality and provide the required service to user; ii) platform independence, i.e., RFBs can be realized via software functions, and can run on several hardware solutions; and, iii) high flexibility and performance, thanks to the fact that RFBs can be deployed where and when they are really needed (hence the superfluid attribute of the architecture). In this context the main question is if it is possible to efficiently manage a 5G superfluid network based on RFBs. In order to give an answer to this question, a 5G architecture was considered to model the needed components in terms of RFBs and the infrastructure resources in terms of physical nodes and HW features. Then the problem of managing a set of RFBs in order to serve the users of a 5G network with a high definition video distribution service was mathematically formulated. Such a model has been tested on a simple but yet representative case study. The results pointed out that the proposed approach is a first step towards a more comprehensive solution. In order to present the obtained mathematical model, Section 2.3.1 gives a description of the 5G architecture under consideration.

2.3.1 Architecture Description

The 5G network model considered in this work is composed of a set of nodes, a set of links, and a set of users. The nodes are used to deploy either small cells, macro cells, or to realize the core network elements of the so called Evolved Packet Core (EPC). Each node is connected to the rest of the network by means of a path of physical links. Each user can be connected to the network by means of a cell (either a macro cell or a small cell). For simplicity, the EPC elements are collapsed in a single site in the model proposed herein. Figure 2.3 reports an example of the considered physical system infrastructure, which is composed of different small cell sites, one macro cell site and one EPC site. In this scenario, each site corresponds to a 5G node. Figure 2.3 reports also the coverage areas of the cells (which are represented by hexagonal layouts for the sake of simplicity). The service area, i.e., the area where the users are located, is assumed to be overlapped with the coverage area of the macro cell. Each 5G node is able to host different RFBs.
An RFB performs specific tasks in the network architecture, such as processing the video to users, or performing networking and physical layer tasks. In addition, each RFB consumes an amount of physical resources on the hosting 5G node. As physical

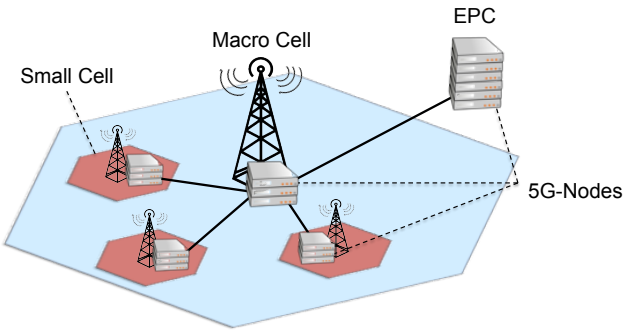


Figure 2.3. Example of a 5G physical system infrastructure

resources the processing capacity (that will be simply denoted as capacity further on) and the memory occupation (in short denoted as memory) have been considered. The following RFBs types are taken into consideration in this work:

- Mobile Edge Computing (MEC) RFB;
- Base Band Unit (BBU) RFB;
- Resource Radio Head (RRH) RFB.

The MEC RFB module is responsible for providing the HD video distribution service to users. A practical example of a MEC RFB is a cache serving a set of videos to users. In general, this module is able to serve an amount of traffic, and consequently a subset of the users spread over the service area. Clearly, the maximum amount of traffic that can be served depends on the amount of resources that are made available to the RFB by the physical node hosting it. The BBU RFB module acts as an interface between the MEC RFB and the RRH RFB. Specifically, the BBU RFB exchanges an amount of IP traffic with the MEC module, and a baseband signal with the RRH one. Similarly to the MEC case, also this module is characterized by an amount of consumed resources to provide the RFB functionality. Eventually the RRH RFB module performs physical layer operations. Specifically, the RRH module handles a set of Radio Frequency (RF) channels with users and the corresponding baseband channels with the BBU RFB. The amount of resources required by this module depends on the type of deployed cell (either a small cell or a macro cell). In this context, the RFBs are organized in logical chains. Specifically, each MEC RFB is logically connected to a BBU RFB, which, in turn, is connected to a RRH RFB and consequently to a set of users.
Figure 2.4 reports an example of RFBs chain and the exchanged information between the modules and the users. In addition, the connection between a pair of RFBs in the chain can be direct, i.e., both RFBs are located on the same physical 5G node, or indirect, i.e., the RFBs are located on two separate nodes. Eventually, RRH RFBs are able to setup a radio link with users, by exploiting the Multi User Multiple

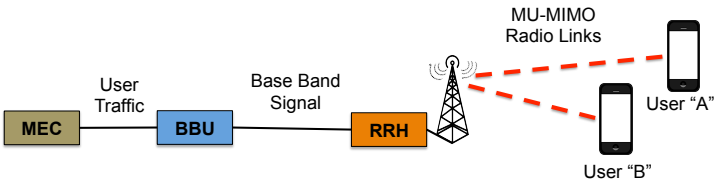


Figure 2.4. An example of RFBs chain

Input Multiple Output (MU-MIMO) technology. Focusing then on the placement of RFBs in the 5G nodes, the RRH RFBs can be placed only in nodes connected to the antennas of the Radio Access Network (RAN). On the contrary, BBU RFBs can be pooled in other nodes. Eventually, MEC RFBs can be potentially deployed in every node of the network. The key feature of the considered 5G system is that the RFBs are fully virtualized resources. Specifically, the RFBs can be dynamically moved across the nodes to satisfy the Key Performance Indicators (KPIs) of the network operator. In Section 2.3.2 the mathematical model built on this architecture is described.

2.3.2 5G Model

We assume that each node is composed of Dedicated HardWare (DHW) and Commodity HardWare (CHW). More in depth, the DHW part hosts RFB functionalities requiring intensive and HW specific operations. These operations include the RRH functions and the BBU functions involving base band processing tasks. On the other hand, the CHW part of the node is used to host RFB functionalities requiring basic processing tasks (i.e., processing of IP packets, processing of video traffic), which are performed by the MEC RFBs and the processing functions of BBU RFBs. Each RFB then consumes an amount of physical resources on the hosting 5G node. Focusing on DHW, we assume that the RFBs require purely capacity resources, while as regards the CHW part of the node, we assume that the resources required by RFBs are constrained by the maximum utilization of the CPUs and the memories. Thus, given the users positions in the considered scenario, the 5G nodes positions, the video requirements, the sets of RFBs, and the RFBs features, the goal is to maximize different KPIs, subject to RFBs placement constraints, 5G node capacity constraints, user coverage constraints and user data constraints.
More formally, let N be the set of nodes and let U be the set of users. In addition, the following sets are introduced: i) set of MEC RFBs types $\mathcal{K}^{MEC}$, ii) set of BBU RFBs types $\mathcal{K}^{BBU}$, iii) set of RRH RFBs types $\mathcal{K}^{RRH}$. The model is presented first reporting the constraints related to RRH RFBs, then the BBU and MEC RFBs constraints and eventually the constraints of the 5G nodes.

$$max \sum_{i,j} t_{i,j} \tag{2.36}$$

The first KPI considered in the model is the maximization of user throughput, that is the maximization of the user performance.

$$\sum_i u_{i,j} = 1 \quad \forall j \tag{2.37}$$

Constraints (2.37) impose that each user has to be served by one and only one 5G node, where the binary variable u_{ij} takes value 1 if the user $j \in U$ is served by node i, 0 otherwise. A user j can be served by node i only if one RRH RFB of type $k \in \mathcal{K}^{RRH}$ installed at node i is able to cover user j:

$$u_{i,j} \leq \sum_k COV_{ijk} r_{ki} \quad \forall i, j \tag{2.38}$$

where COV_{ijk} is a binary input parameter taking value 1 if user j is covered by one RRH RFB of type $k \in \mathcal{K}^{RRH}$ installed on node i (0 otherwise), and r_{ki} is a binary variable taking value 1 if the RRH RFB of type k is installed on node i (0 otherwise). Constraints (2.38) impose also that one RRH RFB has to be installed at node i if at least one user is connected to node i. Moreover, the number of used RRH RFBs has to be lower than the total number of available RFBs of type k, denoted as N_k^{RRH} . More formally:

$$\sum_i r_{ki} \leq N_k^{RRH} \quad \forall k \tag{2.39}$$

In addition, at most one RRH RFB can be assigned to each node:

$$\sum_k r_{ki} \leq 1 \quad \forall i \tag{2.40}$$

Moreover, when an RRH RFB is installed at node i (i.e.,$r_{ki} = 1$), the number of connected users is bounded by the maximum number of terminals for each RRH type k, which is denoted as U_k^{max}. More formally, the following constraints hold:

$$\sum_j u_{ij} \leq \sum k U_k^{max} r_{ki} \quad \forall i \tag{2.41}$$

Each connected user will then receive an amount of RFB capacity δ_{ikj}^{RRH} by assuming that the RRH RFB of type k is installed on node i. The total capacity δ_{ik}^{RRH} provided by one RRH RFB of type k at node i is then computed as:

$$\delta_{ik}^{RRH} = \sum \delta_{ikj}^{RRH} r_{ki} u_{ij} \quad \forall i, k \tag{2.42}$$

δ_{ik}^{RRH} is then bounded by the maximum capacity that can be handled by the installed RRH RFB:

$$\delta_{ik}^{RRH} \leq R_k^{max} \quad \forall i, k \tag{2.43}$$

Moreover, the user traffic has to be lower than the RFB capacity δ_{ikj}^{RRH} :

$$t_{ij} r_{ki} \leq \delta_{ikj}^{RRH} \quad \forall i, j, k \tag{2.44}$$

where $t_{ij} \geq 0$ is a continuous variable representing the traffic between the node i and the user j. This variable has to be larger than zero only if the user j is assigned to the node i, as guaranteed by constraints (2.45):

$$t_{ij} \leq \mathcal{Q} u_{ij} \quad \forall i, j \tag{2.45}$$

where $\mathcal{Q}$ is a very large constant.
An RFB chain composed by one RRH RFB, one BBU RFB, and one MEC RFB has to be deployed in the network in order to serve the users connected to node i. Let b_{kip} be a binary variable equal to 1 if one BBU RFB of type $k \in \mathcal{K}^{BBU}$ placed at node p is used to serve the RRH RFB at node i, 0 otherwise. If the node i has installed one RRH RFB of type w, then one BBU RFB has to serve it:

$$\sum_k \sum_p b_{ikp} = \sum_w r_{wi} \quad \forall i \tag{2.46}$$

In addition, the number of used BBU RFBs is bounded by the number of available RFBs for each BBU type k, which is denoted as N_k^{BBU}:

$$\sum_i \sum_p b_{kip} \leq N_k^{BBU} \quad \forall k \tag{2.47}$$

Focusing on the MEC RFB case, let m_{kip} be a binary variable equal to 1 if one MEC RFB of type $k \in \mathcal{K}^{MEC}$ placed at node p is used to serve the users connected to the RRH RFB at node i, 0 otherwise. The MEC RFB constraint is then expressed as:

$$\sum_k \sum_p m_{kip} = \sum_w r_{wi} \quad \forall i \tag{2.48}$$

Clearly, the total number of used MEC RFBs is bounded by N_k^{MEC} , which is the number of available MEC RFBs of type k:

$$\sum_i \sum_p m_{kip} \leq N_k^{MEC} \quad \forall k \tag{2.49}$$

Moreover, each RFB chain has to ensure compatibility between the RRH and BBU RFBs:

$$r_{ki} \sum_p b_{wip} \leq O_{kw} \quad \forall i, k, w \tag{2.50}$$

where O_{kw} is a binary input parameter, taking value 1 if a RRH RFB of type k and a BBU RFB of type w are compatible with each others, 0 otherwise. Intuitively, these constraints should prevent the connection of an RRH RFB designed for a macro cell with a BBU RFB designed for a small cell, which may otherwise introduce structural incompatibilities (e.g., not enough resources for the BBU RFB to serve the RRH RFB). Eventually, the total traffic to each user is bounded by the HD video capacity provided by the MEC RFB:

$$t_{ij} \leq \sum_p \sum_k m_{kpi} \delta_k^{MEC} \quad \forall i, j \tag{2.51}$$

Focusing on the constraints related to the 5G nodes, the capacity used by RRH and BBU RFBs has to be lower that the one installed on the DHW part:

$$\sum_k r_{ki} \delta_{ik}^{RRH} + \sum_w \sum_p b_{wpi} \delta_w^{BBU} \leq B_i^{DHW} y_i \quad \forall i \tag{2.52}$$

where y_i is a binary variable taking value 1 if node i is used, 0 otherwise. Moreover, the CPU utilization of the MEC RFBs installed at node i is computed as:

$$C_i^{MEC} = \sum_k [C_{ik}^{MS} c_{ik} + C_{ik}^{MD} (\sum_p m_{kpi} \sum_j t_{pj})] \quad \forall i \tag{2.53}$$

where C_{ik}^{MS} and C_{ik}^{MD} are the static and dynamic terms to compute the CPU utilization (C_{ik}^{MS} is the static CPU utilization required by the MEC RFB of type k and C_{ik}^{MD} is a constant to transform the traffic from users into dynamic CPU utilization), and c_{ik} is a binary variable, which takes the value one if at least one MEC RFB of type k is assigned to node i, 0 otherwise. Such a variable is set by means of the following constraints:

$$\sum_{p} m_{kpi} \leq \mathcal{M} c_{ik} \quad \forall i, k \tag{2.54}$$

$$\sum_{p} m_{kpi} + e_{ik} \geq 1 \quad \forall i, k \tag{2.55}$$

$$e_{ik} + c_{ik} = 1 \quad \forall i, k \tag{2.56}$$

where M is a very large constant, and e_{ik} is a binary variable that is equal to 1 when no MEC RFB of type k is assigned to node i, 0 otherwise. The reason for introducing the last two constraints is for assuring that c_{ik} is strictly set equal to zero when no MEC RFB of type k is installed in the node. In this way, in fact, the static amount of capacity C_{ik}^{MS} is not counted. Similarly, the amount of CPU consumed by BBU RFBs is computed as:

$$C_i^{BBU} = \sum_{k} [C_{ik}^{BS} d_{ik} + C_{ik}^{BD} (\sum_{p} b_{kpi} \sum_{j} t_{pj})] \quad \forall i \tag{2.57}$$

where C_{ik}^{BS} and C_{ik}^{BD} are the static and dynamic terms to compute the CPU utilization required by the BBU RFB of type k and d_{ik} is a binary variable, which is computed in a similar way as in the MEC case.

$$\sum_{p} b_{kpi} \leq \mathcal{M} d_{ik} \quad \forall i, k \tag{2.58}$$

$$\sum_{p} b_{kpi} + f_{ik} \geq 1 \quad \forall i, k \tag{2.59}$$

$$f_{ik} + d_{ik} = 1 \quad \forall i, k \tag{2.60}$$

where f_{ik} is a binary variable that is equal to 1 if no BBU RFB of type k is assigned to the node i, 0 otherwise. The total amount of used CPU resources on the CHW part is then bounded by the maximum number of CPU resources:

$$C_i^{MEC} + C_i^{BBU} \leq C_i^{CHW} y_i \quad \forall i \tag{2.61}$$

Focusing on the memory resources, the amount of memory consumed by the MEC RFBs is expressed as:

$$M_i^{MEC} = \sum_{k} [M_{ik}^{MS} c_{ik} + M_{ik}^{MD} (\sum_{p} m_{kpi} \sum_{j} u_{pj})] \quad \forall i \tag{2.62}$$

where M_{ik}^{MS} is the static memory utilization required by a MEC RFB of type k and M_{ik}^{MD} is a constant to obtain the dynamic memory utilization, given the number of

connected users. Moreover, the amount of memory consumed by the BBU RFBs is defined as:

$$M_i^{BBU} = \sum_k [M_{ik}^{BS} d_{ik} + M_{ik}^{BD} (\sum_p b_{kpi} \sum_j u_{pj})] \quad \forall i \tag{2.63}$$

where M_{ik}^{BS} and M_{ik}^{BD} are the static and dynamic terms to compute the memory consumed by a BBU RFB of type k. The total amount of used memory resources is then bounded by the maximum number of memory resources:

$$M_i^{MEC} + M_i^{BBU} \leq M_i^{CHW} \quad \forall i \tag{2.64}$$

The same set of constraints has been implemented also adding another KPI, that is the minimization of the number of used nodes.This objective aims to: i) limit the operating expenditures (OPEX) paid by operator (e.g., the node energy costs or the management ones), ii) efficiently exploit the nodes that are used. Such objective is expressed as follows:

$$\min \sum_i y_i \tag{2.65}$$

The latter case has been solved by means the ϵ-constrained method (described in Section 3.2.1). In this context such a method consists in maximizing the user throughput, limiting the number of used nodes by adding the other objective among the constraints as follows:

$$\sum_i y_i \leq N_{used}^{max} \tag{2.66}$$

where N_{used}^{max} is the maximum number of used nodes, which is varied between 1 and $|\mathcal{N}|$. Section 2.3.3 reports the scenario description on which the presented model has been tested and the results obtained.

2.3.3 Performance Evaluation

A scenario composed of one macro cell, four small cells, and 260 users requesting 5G services has been considered. Figure 2.5 reports the cells and the user positions. More in depth, the macro cell is placed in the center of the service area. Each small cell is placed at a distance of 120 [m] from the macro cell. It is assumed that small cells may interfere with each others, while the central macro cell may interfere with a set of neighboring macro cells, placed at the corners of a square centered by the considered macro cell, with an edge equal to 1000 [m]. Focusing on users, 70% of them are randomly deployed over the whole service area, while 30% are generated in a circle of radius equal to 50 [m] centered in each small cell (thus justifying the small cell deployment).

Focusing on the RFBs, it is assumed a total of 5 RRH RFBs, 5 BBU RFBs, and 5 MEC RFBs. In addition, two types of RRH RFBs, two types of BBU RFBs, and one type of MEC RFB are assumed. The intuition of having two types of RRH RFBs and BBU RFBs relies on the fact that the traffic handled by the macro cell node is in general higher than the one of a small cell. Therefore, the resource requirements of the associated RFBs may be different, resulting in two different RFB types. As regards the setting of the rest of parameters the reader is referred to

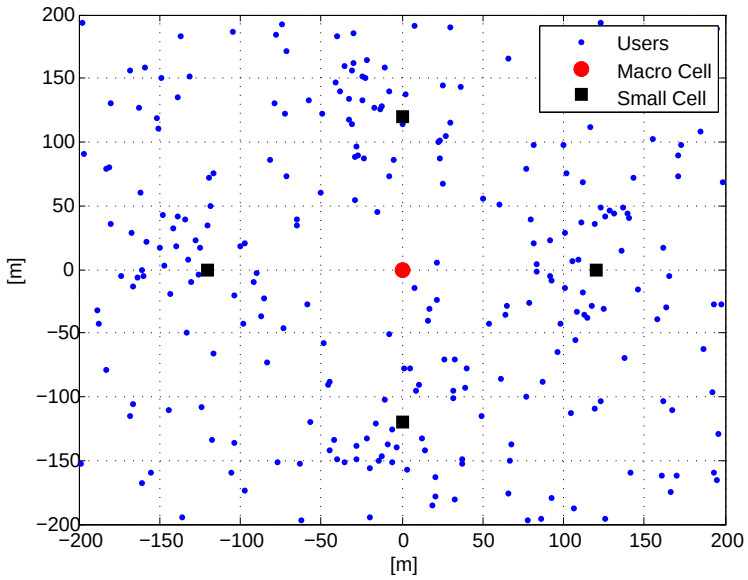


Figure 2.5. Spacial distribution of users and cells defined for the selected scenario

[23]. Moreover the following assumptions have been adopted: i) the network has to satisfy the amount of traffic generated by users with the RFBs deployed in the nodes; ii) the resources of each small cell node are set to host at least one RRH RFB and one BBU RFB for the DHW, and one BBU RFB and one MEC RFB in the CHW; iii) the macro cell node and the EPC node are designed to pool the BBU and MEC RFBs from the small cells; and, iv) an amount of spare resources is always reserved in each node (i.e., to cope with future traffic increases). The proposed optimization model has been solved over the considered scenario on a high performance computing cluster, composed of four nodes, each of them with 32 cores and 64 GB of RAM, for a total computing power of around 1.5 TeraFlops/s using CPLEX solver (version 12.6). In addition to the peak traffic condition, the case in which only 10% of users generate traffic has been taken into account. Such a scenario is identified as off-peak condition. The goal is in fact to assess the performance of the considered architecture under different traffic conditions. We initially take into account the amount of traffic t_{ij} served to each user.

More in depth, Figure 2.6 reports the Cumulative Distribution Function (CDF) of the user traffic for the two traffic conditions. The figure reports also the CDFs for the following cases: i) best node allocation policy, i.e., $t_{ij} = max_i \delta_{ikj}^{RRH}$; and, ii) worst node allocation policy, i.e., $t_{ij} = min_i \delta_{ikj}^{RRH}$. Interestingly, the traffic served to users during the peak traffic condition is close to one achieved by the best node allocation, with an average traffic of more than 100 [Mbps] per user. However, a small subset of users (around 10%) is experiencing very low traffic (i.e., close to 0). By further investigating this issue, it has been found that such users are close to the edges of the macro cell, i.e., they are the ones experiencing the worst channels conditions. To overcome this issue, these users could be potentially covered also by the neighboring macro cells in a real environment.

In addition, Figure 2.6 reports also the CDF when the off-peak traffic condition is considered. In this case, the average traffic per user is still higher than 80 [Mbps],

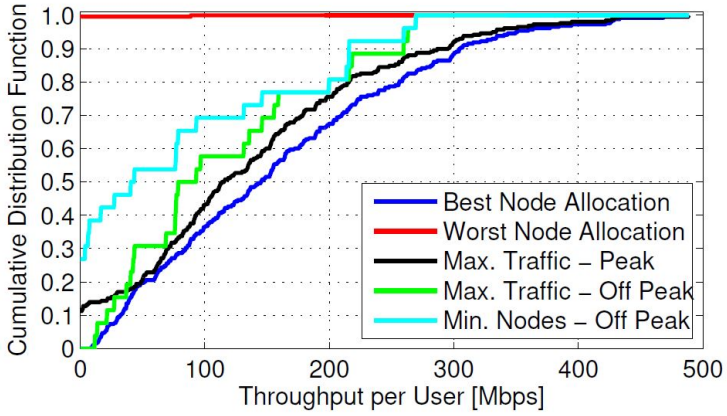
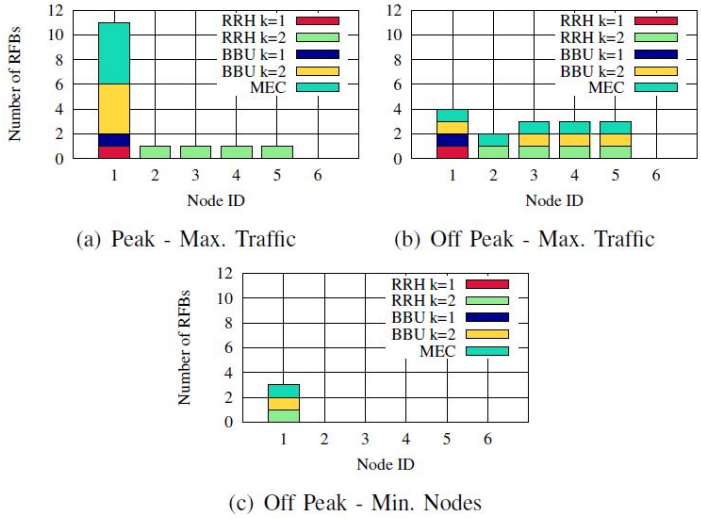


Figure 2.6. Cumulative Distribution Function (CDF) of the user traffic



(a) Peak - Max. Traffic (b) Off Peak - Max. Traffic

(c) Off Peak - Min. Nodes

Figure 2.7. RFBs placement over the set of nodes for different traffic conditions

while no user exhibits extremely low traffic conditions. Eventually, the figure reports the CDF of the user throughput for the off-peak traffic condition when the minimization of used nodes is pursued (i.e., by adopting the ϵ-constrained method). In this case, the users are connected to a single node in the network, i.e., the macro cell. Clearly, this choice has an impact on the throughput, which tends to be decreased (i.e., the average throughput is less than 50 [Mbps]). Figure 2.7 reports the RFBs placement over the set of nodes. Focusing on the peak traffic condition and the maximization of the user throughput (Figure 2.7(a)), all the RRH RFBs are exploited, in order to maximize the performance to users. In addition, the BBU and MEC RFBs are all located on the macro cell node. Then, Figure 2.7(b) reports the off-peak traffic. In this case, the BBU and MEC RFBs tend to be spread over the nodes. This is due to the fact that the number of used nodes is not taken into account when the objective function is solely the maximization of the users traffic. As a result, all the nodes may be potentially exploited even if the number of users is

low. Focusing on the same traffic condition and on the minimization of the number of used nodes (Figure 2.7(c)), the macro cell node is hosting one RRH RFB of type 2, one BBU RFB of type 2 and one MEC RFB. In this last part we analyze the utilization of nodes resources consumed by the RFBs when the maximization of the user throughput is pursued.

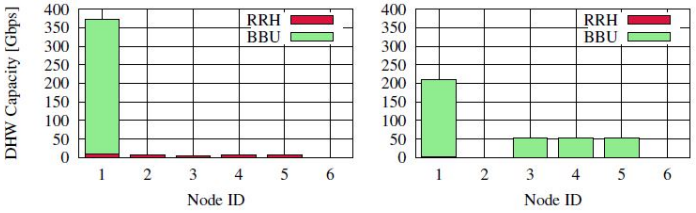


Figure 2.8. DHW capacity across the set of nodes

Focusing on the DHW part, Figure2.8(a) reports the amount of used capacity for the peak traffic condition. As expected, the largest amount of capacity is consumed by the BBU RFBs, while the RRH RFBs marginally impact the overall capacity. This is due to the fact that BBU RFBs perform the baseband operations, which are pretty intensive on the computing resources of the node. Moreover, the total capacity consumed on the macro cell node is more than 350 [Gbps], i.e., close to 50% of the installed capacity. On the contrary, the small cell nodes are lightly utilized. Moreover, Figure 2.8(b) presents the results for the off-peak traffic condition. Eventually, the amount of capacity consumed on the small cell nodes tends to promptly increase, as the BBU RFBs are deployed also on most of them. Nevertheless, the amount of capacity used on the macro cell node is still higher than 200 [Gbps] (i.e., more than 20%).
In conclusion these results show that: i) users are able to achieve very good throughput in the downlink direction; ii) the RFBs placement is impacted by the number of users and the considered strategy; iii) the BBU RFBs of the same type and the MEC RFBs may be efficiently pooled on the same node to better exploit the CHW and DHW resources.

2.4 Inspired research

In this chapter some emerging optimization problems on telecommunication networks have been described, mathematically formulated and solved on a real scenario and, in the latter case, on a realistic simulated scenario. Each one of the problems considered have been modeled or, let us say, "reduced" to a single objective mixed integer linear programming problem even thought they have a multi-objective nature. Indeed, focusing on the optimization criteria, in each of the different problems under consideration, two objectives have to be aggregated in order to validate the model and obtain some solutions of practical importance for the real case. In the first problem two different aspects have been considered: the energy efficiency and the

network reliability. According with the considerations given in the introduction to this problem, these two criteria are in conflict with each other and this means that a solution, which is optimal for one of the two objectives, could be not optimal for the other one. On the contrary, several trade-off solutions may exist instead of a unique optimal solution. In the second problem, two different objectives have been included: electricity and lifetime costs and users QoS. Also in this case, these criteria may generate a set of efficient solutions. For the latter problem, as described in Section 2.3, two different KPIs have been analyzed: the maximization of the users throughput and the minimization of the number of used nodes. These two criteria are clearly in conflict and in this case, a multi-objective approach has been adopted in order to solve the problem, that is the ϵ-constrained method. Such a method will be described in details in the following chapter. In general, this method is not efficient because it consists in adding knapsack constraints to the problem formulation, keeping only one of the objectives. This change makes the formulation more complex and often NP-hard even if the original problem is not NP-hard.
Focusing on the formulations previously presented, they are very complex and for this reason the use of general solution approaches for multi-objective integer (or mixed integer) programming, like the ϵ-constrained method, may be very inefficient. From this observation, it arises the necessity to design ad hoc algorithms in order to deal with their multi-objective nature and to explore their set of efficient solutions. In fact, the knowledge of a complete set of efficient solutions of a multi-objective problem is very important in order to provide a useful decision support tool to the decision maker.
In literature, as shown in the following chapter, several solution approaches for multi-objective programming have been developed, but the research in this field does not yet keep up with the complexity of the formulations derived from real problems like those presented in this chapter. For such a reason in this thesis two bi-objective problems, that are a starting point for more complex formulations, have been considered: the bi-objective integer min cost flow and the bi-objective spanning tree problems. Also with reference to the telecommunication sector, these problems represent the basic formulations for the efficient configuration and management of the TLC networks. A general two-phase framework is proposed in order to determine a complete set of efficient solutions for such problems. This work represents only the first step in the direction of reducing the existing gap between the complex formulations that arise from the real world applications and the current state of the art of the solution approaches.

Chapter 3

Multicriteria optimization

In decision making processes in complex organizational contexts it is often insufficient to make a decision based on a single criterion and it is more realistic to consider different goals simultaneously, often in conflict with each other. Indeed, in many real world problems, different points of view have to be taken into account. Several examples of that appear in different application fields. In the transportation sector, efficiency and cost-effectiveness on the one hand and security on the other hand, in the telecommunications sector, energy saving and reliability should be combined, in the construction industry cost and environmental impact, while in finance expected return and risk have to be considered simultaneously. From the necessity to deal with this type of problems the development of multi-objective optimization and a new concept of optimal solution follow. Indeed, when considering more objectives (criteria) in conflict, it is not guaranteed that a solution which is optimal for one of the criteria is also optimal for the other objectives and it is no longer possible to use the same concept of optimality as for the single objective case. Then the definition of Pareto optimal solution or efficient solution is introduced. Such a solution can not be improved according to a criterion without getting worse at least one of the others. Generally a multi-objective optimization problem, because of the conflictive relation between criteria, will have more than one efficient solution, and among them the decision maker can choose the one that he or she prefers. The basic form of multi-objective optimization is multiple objective linear programming, characterized by objectives and constraints which are linear functions and it is possible to distinguish between the continuous and the integer case. For multi-objective linear programming it was developed also a duality theory (see [50]), that is useful for designing dual solution methods (see [34]). Different solution approaches can be considered distinguishing between continuous and integer case. Moreover, a classification of the solution methods for multi-objective programming, more theoretical than practical, is based on the role of the decision maker in the resolution process. In this context it is possible to distinguish among "a priori mode", "a posteriori mode" and "interactive mode". In the first case all the preferences are known at the beginning of the decision making process. In the second case the set of all efficient solutions is generated and then analyzed according with the decision maker's preferences. In the last case the preferences are introduced by the decision maker during the resolution process and used to drive the search for a satisfying

compromise.
In the next sections, definitions, concepts and solution approaches for multi-objective linear programming in the continuous and integer cases are recalled. An introduction to the duality theory for multi-objective linear programming is given in section 3.1.2.

3.1 Multiobjective linear programming

A multi-objective linear programming problem can be formulated as follows:

$$\min Cx$$
$$Ax = b$$
$$x \geq 0$$

where $x \in \mathbb{R}^n$ is the vector of decision variables, $C \in \mathbb{R}^{p \times n}$ is the coefficient matrix defining the p criteria of the problem, $A \in \mathbb{R}^{m \times n}$ is the coefficient matrix and $b \in \mathbb{R}^m$ is the vector defining the right hand side of the problem constraints. In the context of multi-objective optimization it is useful to distinguish the feasible set in the decision space (or decision set) $X = \{x \in \mathbb{R}^n : Ax = b, x \geq 0\}$, consisting of feasible solutions, and 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.

Example 1. *This example is a bi-objective linear programming problem in two variables.*

$$\max(-x_1 + 2x_2, 2x_1 - x_2)$$
$$x_1 + x_2 \leq 7$$
$$-x_1 + x_2 \leq 3$$
$$x_1 - x_2 \leq 3$$
$$x_1,\ x_2 \geq 0$$
$$x_1,\ x_2 \leq 4$$

Figure 3.1 shows the decision space of this problem and figure 3.2 reports the corresponding objective space.

A fundamental concept in multi-objective programming is the definition of Pareto optimal solution or efficient solution. Indeed, differently from the single objective case, characterized by a complete order on the outcome set, when multiple objectives are involved, this set is only partially ordered (see [31]). A new definition of optimality is introduced, based on the concept of Pareto dominance as defined in Definition 24.

Definition 24 (Pareto dominance)**.** *A feasible solution $x \in X$ is dominated by another feasible solution $x' \in X$ if $Cx' \leq Cx$ with strict inequality for at least one of the objectives.*

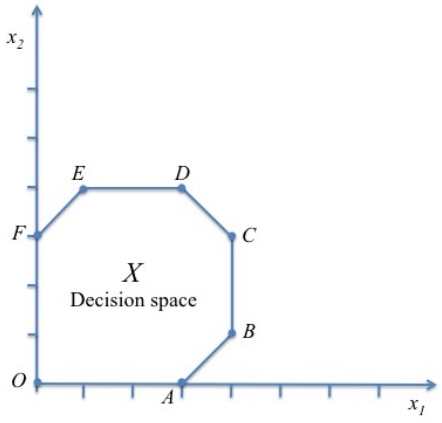


Figure 3.1. Decision space for Example 1

Definition 25 (Efficiency or Pareto optimality)**.** *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 26 (Extreme efficient solution)**.** *An efficient solution which defines an extreme point of conv(Y) is called extreme efficient solution.*

Definition 27 (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 28 (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 29 (Pareto or non-dominated frontier)**.** *The set of non-dominated vectors Y_N is called Pareto or non-dominated frontier.*

According with the previous definitions, solving a multi-objective linear programming problem means finding the set Y_N or a complete set of efficient solutions denoted by X_E.
The main result is the fundamental theorem of multiple objective linear programming (see [55]).

Theorem 7 (Fundamental theorem of multiple objective programming)**.** *A feasible solution $x^* \in X$ is efficient if and only if there exists $\lambda > 0$ such that $\lambda^T Cx^* \leq \lambda^T x \quad \forall x \in X$.*

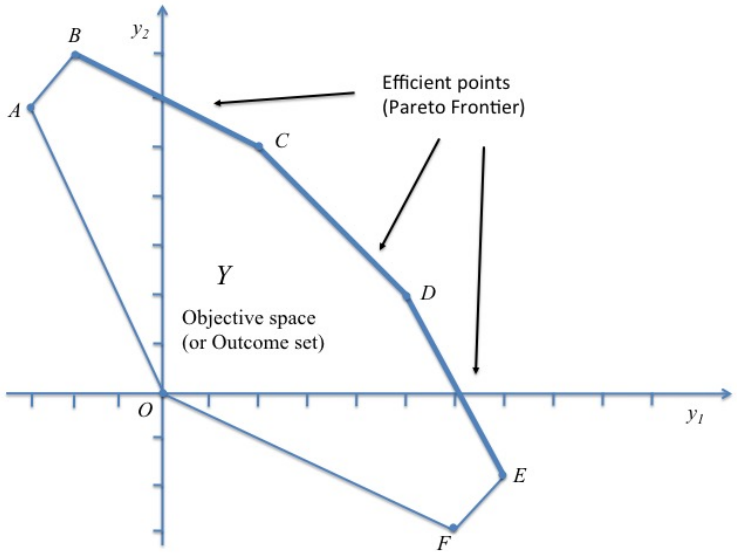


Figure 3.2. Objective space for Example 1

As regards solution methods for multi-objective linear programming, three main classes can be identified. The first one is related to the simplex algorithms, that are extensions of simplex algorithm for the single objective case to deal with more objectives. Because of the possibility to have a large number of efficient extreme solutions, the algorithms of this class can be very slow. The second class consists of interior point algorithms which, at the current state of the art, do not find all efficient solutions. The third class includes algorithms that are specific for multi-objective problems. Such algorithms work in the outcome set of the problem for finding non-dominated points. Three main motivations can be identified to prefer the exploration of the outcome set rather than the exploration of the decision set. The first reason is that the dimension of the outcome set is usually smaller than the dimension of the decision space. This implies that less computational effort is required for computing the non-dominated points. The second reason is related to the decision maker. Indeed, in many practical cases it is not possible for the decision maker to choose a solution among the huge number of efficient solutions. Eventually it appears to be easier and more natural to compare the objective values rather than the decisions leading to them. In this last class of solution algorithms it is possible to identify a category represented by the dual methods. This is still a relatively new approach, yet a first attempt to use duality results for designing a dual algorithm for multiple linear programmes with the goal to speed-up the computation times was made by Ehrgott et al. in [34]. Such an algorithm, depending on the problem structure, can be faster than its primal version and it will be presented in Chapter 5. In section 3.1.1 the primal version of Benson's algorithm is presented. Preliminary notions and duality results needed to introduce its dual variant will be reported in section 3.1.2.

3.1.1 Solving MOLP in Objective Space: Bensons's algorithm

As mentioned above there exist at least three reasons to prefer the generation of the non-dominated points in the outcome set Y instead of the efficient solutions in the decision space. The first one is related to the dimension: usually Y_N has a much simpler structure and smaller dimension than X_E. Indeed $Y_N \subseteq \mathbb{R}^p$ and $X_E \subseteq \mathbb{R}^n$ where p is usually much smaller than n. This implies that in general generating all or portions of Y_N requires less computational effort than generating all or portions of X_E. The second reason regards the decision maker: in practice it was shown that the decision maker prefers basing his or her choice looking at the points in Y_N rather than in X_E. Eventually, another reason is the existence of solutions in X_E that are mapped by C onto a single point in Y_N. Thus generating points directly from Y_N avoids the possibility of redundant calculations of solutions from X_E not useful for the decision maker. In this context the first algorithm able to generate the set of all efficient extreme points in the outcome set, was proposed by Benson (see [11]) and it is called Outer Approximation Algorithm. This algorithm is based on some theoretical results that we will report. The main idea is to consider another polyhedron $\bar{Y}$, opportunely defined, with the characteristic to have the same set of non-dominated points of Y, for which it is easier to identify the set of all efficient extreme points. In the following the main theoretical results on which the algorithm is based are reported (see [11] for more details and proofs).

Prerequisities

Assuming to have a multi-objective linear programming problem where p criteria (with rank of matrix C equal to $q \geq 1$) have to be maximized over X (supposed nonempty and compact), it is possible to prove Proposition 2.

Proposition 2. *The dimension of Y satisfies $dim Y \leq q$.*

The anti-ideal outcome for a multi-objective linear programming problem where p criteria have to be maximized over X, is defined as:

$$y_i^{AI} = \min_{y \in Y} y_i \quad \forall i = 1, ..., p$$

Let $\hat{y} \in \mathbb{R}^p$ satisfy $\hat{y} < y^{AI}$. The polyhedron mentioned before, that is instrumental in the algorithm, is defined as follows:

$$\bar{Y} = \{y \in \mathbb{R}^p : \hat{y} \leq y \leq Cx \text{ for some } x \in X\} \tag{3.1}$$

It is possible to prove that this set is nonempty, bounded in $\mathbb{R}^p$ and of dimension p. Denoting with $\bar{Y}_N$, the set of non-dominated points of $\bar{Y}$, the first important result on which is based the algorithm is the following:

Theorem 8. $Y_N = \bar{Y}_N$.

In the Outer Approximation Algorithm, an initial simplex containing $\bar{Y}$ is built.

This construction is based on the following results:

Theorem 9. *Let $\beta = max\langle e, y\rangle$ with $y \in \bar{Y}$ and $e \in \mathbb{R}^p$ vector of entries equal to 1. Let $v^0 = \hat{y}$ and for each $j = 1, ..., p$ let $v^j \in \mathbb{R}^p$ be defined by:*

$$v_i^j = \begin{cases} \hat{y}_i & \text{if } i \neq j \\ \beta + \hat{y}_j - \langle e, \hat{y}\rangle & \text{if } i = j \end{cases}$$

The convex hull S of $V(S) = \{v^j \mid j = 0, ..., p\}$ is a p-dimensional simplex with vertex set $V(S)$ and S contains Y.

Theorem 10. *The simplex S defined in the previous theorem may also be written as:*

$$S = \{y \in \mathbb{R}^p \mid \hat{y} \leq y, \ \langle e, y\rangle \ \leq \beta\}$$

As the set $\bar{Y}$ is nonempty and bounded in $\mathbb{R}^p$, it follows that if a point $\bar{p} \in \text{int } \bar{Y}$ is chosen, starting from the simplex S defined in Theorem 9 , the algorithm will iteratively generate a finite number of nonempty, compact, polyhedra S^k, $k = 0, 1, ...K$ such that $S = S^0 \supset S^1 \supset ... \supset S^{K-1} \supset S^K = \bar{Y}$. At the generic iteration k, a vertex $y^k \in S^k$ will be identified such that $y^k \notin \bar{Y}$. Subsequently the unique point w^k on the boundary of Y that lies on the line segment joining $\bar{p}$ with y^k will be identified. Indeed, the next result implies that w^k belongs to the weakly non dominated set of $\bar{Y}$, denoted by $\bar{Y}_{WE}$.

Theorem 11. *Let $\bar{p} \in int\ \bar{Y}$ and suppose that $y \geq \hat{y}$ and $y \notin \bar{Y}$. Let w denote the unique point on the boundary of $\bar{Y}$ that belongs to the line segment connecting y and $\bar{p}$. Then $w \in \bar{Y}_{WE}$.*

It is possible to prove that $\bar{Y}$ is a p-dimensional polyhedron with a finite number of faces and that a set $F \subseteq \mathbb{R}^p$ is a face of $\bar{Y}$ if and only if F coincides with the optimal solution set $Y^*(\alpha)$ for the following problem:

$$\max\langle \alpha, y\rangle$$

$$y \in \bar{Y}$$

for some $\alpha \in \mathbb{R}^p$. From the definition of $\bar{Y}$ and the fact that $\hat{y} < Cx \quad \forall x \in X$ it follows that p of the $p-1$-dimensional faces of $\bar{Y}$ are given by:

$$F_j = \{y \in \bar{Y} \mid y_j = \hat{y}_j\} \quad \forall j = 1, ..., p$$

Moreover for each $j = 1, .., p$ either $F_j \subseteq \bar{Y}_{WE}$ or ri $F_j \cap \bar{Y}_{WE} = \emptyset$, where ri F_j is the relative interior of F_j. Indeed $\hat{y} \in F_j \quad \forall j = 1, ..., p$, and $\hat{y} \notin Y_{WE}$. From these observations, it derives that the point w in the last theorem lies in some face $F \subseteq \bar{Y}_{WE}$ of $\bar{Y}$ that satisfies $F \neq F_j \quad \forall j = 1, ..., p$. Any such face F coincides with the optimal solution set $Y^*(\alpha)$ for some $\alpha \in \mathbb{R}^p$ such that $\alpha > 0, \ \alpha \neq 0$. For finding such a face F the following result is needed.

Theorem 12. *Assume that $w \in \bar{Y}_{WE}$, and let (u^{*T}, v^{*T}) denote any optimal solution for the dual linear program of the problem Q_w:*

$$\max t$$
$$Cz - et \geq w \tag{3.2}$$
$$Az = b \tag{3.3}$$
$$z, t \geq 0$$

where $u^* \in \mathbb{R}^p$ and $v^* \in \mathbb{R}^m$ correspond to constraints 3.2 and 3.3, respectively, of the problem Q_w. Then $u^* \geq 0$, $u^* \neq 0$, and w belongs to the weakly efficient face $Y^*(u^*)$ of $\bar{Y}$. Furthermore, $Y^*(u^*)$ is given by:

$$Y^*(u^*) = \{y \in \bar{Y} \mid \langle u^*, y\rangle = \langle b, v^*\rangle\}$$

The previous theorem provides the basis for building some linear inequality cuts needed in the Outer Approximation Algorithm as showed in the following scheme.

Algorithm scheme

Initialization Compute a point $\bar{p} \in$ int $\bar{Y}$ and build the p-dimensional simplex $S^0 = S$ containing $\bar{Y}$ defined in (3.1). Store both the vertex set $V(S^0)$ of $S^0 = S$ and the inequality representation of $S^0 = S$. Set $k = 0$ and go to iteration k.

Iteration k $k \geq 0$. See steps $k1 - k4$.

Step k1 If $\forall y \in V(S^k$, $y \in \bar{Y}$ is satisfied, then stop. $Y = S^k$. Otherwise, choose any $y^k \in V(S^k)$ such that $y^k \notin \bar{Y}$ and continue.

Step k2 Find the unique value λ_k of λ, $0 < \lambda < 1$, such that $\lambda y^k + (1-\lambda)\bar{p}$ belongs to the boundary of $\bar{Y}$, and set $w^k = \lambda_k y^k + (1 - \lambda_k)\bar{p}$.

Step k3 Set $S^{k+1} = S^k \cap \{y \in \mathbb{R}^p \mid \langle u^k, y\rangle \leq \langle b, v^k\rangle\}$, where (u^{kT}, v^{kT}) is any dual optimal solution to the linear program Q_w with $w = w^k$.

Step k4 Using $V(S^k)$ and the definition of S^{k+1} given in the previous step, determine $V(S^{k+1})$. Set $k = k + 1$ and go to iteration k.

Each of the inequalities $\langle u^k, y\rangle \leq \langle b, v^k\rangle \quad \forall k \geq 0$ appended to S^k is called an inequality cut and it is constructed so that S^{k+1} cuts off a portion of S^k containing y^k in such a way that $S^k \supset S^{k+1} \supset \bar{Y}$. The algorithm proposed by Benson is able to generate all the efficient extreme points in the outcome set in a finite number of iterations as reported in Theorem 13.

Theorem 13. *The Outer Approximation Algorithm is finite and, when it terminates, $S^k = \bar{Y}$, where $K \geq 0$ is the final iteration number.*

When the algorithm terminates, the set of all efficient extreme points in the outcome set Y for the original multiple objective linear program, can be easily found by means of Theorem 14.

Theorem 14. *Let $K \geq 0$ denote the iteration number in which $S^k = \bar{Y}$ and the Outer Approximation Algorithm terminates. Let*

$$E = \{y \in V(S^k) \mid y > \hat{y}\}.$$

Then E is identical to the set of all efficient extreme points of Y.

An improvement of this algorithm was proposed in [34], where it is not necessary to work with bounded simplices, as in the original version, and check whether a vertex is non-dominated or not, is superfluous. In the same paper a dual variant of Benson's algorithm was proposed. In order to explain this variant, an introduction to duality theory for multi-objective linear programming is presented in Section 3.1.2.

3.1.2 Duality

In this section the main concepts and results of duality theory for multiple objective linear programs are reported. This theory is based on the duality relation between the polyhedral image set of the primal problem (P) and the polyhedral image of the dual problem (D). Indeed in [50], Heyde and Löhne, show that, there exists an inclusion reversing one-to-one map between the minimal faces of the primal outcome set and the maximal faces of the dual outcome set.

Preliminaries

Let $\mathcal{C} \subseteq \mathbb{R}^p$ be a closed convex cone. An element $y \in \mathcal{A}$ is called $\mathcal{C}$-minimal if $(\{y\} - \mathcal{C}\backslash\{0\}) \cap \mathcal{A} = \emptyset$ and $\mathcal{C}$-maximal if $(\{y\} + C\backslash\{0\}) \cap \mathcal{A} = \emptyset$. A point $y \in \mathcal{A}$ is called weakly $\mathcal{C}$-minimal if $(\{y\} - \text{ri } \mathcal{C}) \cap \mathcal{A} = \emptyset$ and weakly $\mathcal{C}$-maximal if $(\{y\} + \text{ri } \mathcal{C}) \cap \mathcal{A} = \emptyset$, where ri $\mathcal{C}$ is the relative interior if $\mathcal{C}$. Considering the following two special cones:

$$\mathcal{C} = \mathbb{R}^p_{\geq} = \{x \in \mathbb{R}^p : x_k \geq 0, k = 1, ..., p\}$$

and

$$\mathcal{C} = \mathcal{K} = \mathbb{R}_{\geq} e^p = \{y \in \mathbb{R}^p : y_1 = ... = y_{p-1} = 0, y_p \geq 0\}.$$

We define the weakly $\mathbb{R}^p_{\geq}$-minimal elements of $\mathcal{A}$ (also named set of weakly non dominated points of $\mathcal{A}$) and the set of $\mathcal{K}$-maximal elements of $\mathcal{A}$ as follows:

$$\text{wmin}_{\mathbb{R}^p_{\geq}} \mathcal{A} = \{y \in \mathcal{A} : (\{y\} - \text{int}\mathbb{R}^p_{\geq}) \cap \mathcal{A} = \emptyset\}.$$
$$\text{wmax}_{\mathcal{K}} \mathcal{A} = \{y \in \mathcal{A} : (\{y\} + \mathcal{K}\backslash\{0\}) \cap \mathcal{A} = \emptyset\}.$$

Let $\mathcal{A} \subseteq \mathbb{R}^p_{\geq}$ be a convex set. Some useful definitions are reported in the following.

Definition 30 (Face)**.** *A convex subset $\mathcal{F} \subseteq \mathcal{A}$ is defined a face of $\mathcal{A}$ if for all $y^1, y^2 \in \mathcal{A}$, and $\alpha \in (0,1)$ such that $\alpha y^1 + (1-\alpha)y^2 \in \mathcal{F}$, it holds that $y^1, y^2 \in \mathcal{F}$.*

Definition 31 (Proper face)**.** *A face $\mathcal{F}$ of $\mathcal{A}$ is called proper if $\emptyset \neq \mathcal{F} \neq \mathcal{A}$.*

Definition 32. *A point $y \in \mathcal{A}$ is defined an extreme point of $\mathcal{A}$ if $\{y\}$ is a face of $\mathcal{A}$.*

Definition 33 (Recession direction)**.** *A recession direction of $\mathcal{A}$ is a vector $d \in \mathbb{R}^p$ such that $y + \alpha d \in \mathcal{A}$ for some $y \in \mathcal{A}$ and all $\alpha \geq 0$.*

Definition 34 (Recession cone)**.** *The recession cone $\mathcal{A}_\infty$ of $\mathcal{A}$ is the set of all recession directions $\mathcal{A}_\infty = \{d \in \mathbb{R}^p : y + \alpha d \in \mathcal{A}$ for some $y \in \mathcal{A}$ for all $\alpha \geq 0\}$.*

Definition 35 (Extreme recession direction)**.** *A recession direction $d \neq 0$ is named extreme if there does not exist recession directions $d^1, d^2 \neq 0$ with $d^1 \neq \alpha d^2$ for all $\alpha > 0$ such that $d = \frac{1}{2}(d^1 + d^2)$.*

A polyhedral convex set $\mathcal{A}$ is defined by $\{x \in \mathbb{R}^p : Dx \geq f\}$ where $D \in \mathbb{R}^{m \times p}$ and $f \in \mathbb{R}^m$.

Definition 36 (Supporting hyperplane)**.** *The set $\mathcal{H} = \{y \in \mathbb{R}^p : \lambda^T y = \gamma\}$, where $\lambda \in \mathbb{R}^p$ and $\gamma \in \mathbb{R}$, is a supporting hyperplane to $\mathcal{A}$ if $\lambda^T y \geq \gamma$ for all $y \in \mathcal{A}$ and there is some $y^0 \in \mathcal{A}$ such that $\lambda^T y^0 = \gamma$.*

Proposition 3. *A polyhedral set $\mathcal{A}$ has a finite number of faces.*

Proposition 4. *A subset $\mathcal{F}$ of $\mathcal{A}$ is a face if and only if there are $\lambda \in \mathbb{R}^p$ and $\gamma \in \mathbb{R}$ such that $\mathcal{A} \subseteq \{y \in \mathbb{R}^p : \lambda^T y \geq \gamma\}$ and $\mathcal{F} = \{y \in \mathbb{R}^p : \lambda^T y = \gamma\} \cap \mathcal{A}$.*

Proposition 5. *$\mathcal{F}$ is a proper face if and only if $\mathcal{H} = \{y \in \mathbb{R}^p : \lambda^T y = \gamma\}$ is a supporting hyperplane to $\mathcal{A}$ with $\mathcal{F} = \mathcal{A} \cap \mathcal{H}$.*

Definition 37. *The proper $(r-1)$-dimensional facets of an r-dimensional polyhedral set $\mathcal{A}$ are called facets od $\mathcal{A}$.*

Theorem 15 is due to Rockafeller (see [72]).

Theorem 15. *A polyhedral convex set $\mathcal{A}$ can be represented by a finite set of inequalities and the set of all extreme points and extreme directions of $\mathcal{A}$.*

Let $\xi = \{x^1, ..., x^r, d^1, ..., d^t\}$ be the set of all extreme points and directions of $\mathcal{A}$, then

$$\mathcal{A} = \{y \in \mathbb{R}^p : y = \sum_{i=1}^{r} \alpha_i x^i + \sum_{j=1}^{t} v_j d^j \text{ with } \alpha_i \geq 0, v_j \geq 0, \text{ and } \sum_{i=1}^{r} \alpha_i = 1\}$$

For a polyhedral convex set $\mathcal{A}$, the extreme points are called vertices and the set of all vertices of a polyhedron $\mathcal{A}$ is indicated by vert$\mathcal{A}$.

Geometric duality

The geometric duality for multi-objective linear programming was developed by Heyde and Löhne (see [50]). Let $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$, $P \in \mathbb{R}^{p\times n}$ and $e = \{1, ..., 1\} \in \mathbb{R}^p$. Consider the multiple objective linear programming problem:

$$\text{wmin}_{\mathbb{R}^p_{\geq}} P(\mathcal{X}) \quad , \quad \mathcal{X} = \{x \in \mathbb{R}^n : Ax \geq b\}$$

This problem is the primal problem and consists in finding the weakly non-dominated points of the outcome set $P(\mathcal{X})$. Theorem 7 due to Isermann, assures that a point $y^* \in P(\mathcal{X})$ is a weakly non-dominated point of $P(\mathcal{X})$ if and only if there exists a vector $w \in \mathbb{R}^p_{\geq}$ with $e^T w = 1$ such that $w^T y^*$ is the minimal element of the set $\{w^T y : y \in P(\mathcal{X})\}$. Any x^* such that $Px^* = y^*$ is therefore an optimal solution of the linear programme $\min\{w^T Px : Ax \geq b\}$. The dual multiple objective linear programme according with the geometric duality is given by:

$$\text{wmax}_{\mathcal{K}} D(\mathcal{U}) \quad , \quad \mathcal{U} = \{(u, \lambda) \in \mathbb{R}^m \times \mathbb{R}^p : (u, \lambda) \geq 0,\ A^T u = P^T \lambda,\ e^T \lambda = 1\}$$

where $\mathcal{K} = \{y \in \mathbb{R}^p : y_1 = y_2 = ... = y_{p-1} = 0,\ y_p \geq 0\}$ and $D : \mathbb{R}^m \times \mathbb{R}^p \to \mathbb{R}^p$ given by:

$$D(u, \lambda) = (\lambda_1, ..., \lambda_{p-1}, b^T u)^T$$

The dual problem consists in finding the $\mathcal{K}$-maximal elements of $D(\mathcal{U})$. The vector λ corresponds to the weight vector w in the weighted sum linear programme $\min\{w^T Px : Ax \geq b\}$. Considering a fixed $\lambda \geq 0 : e^T \lambda = 1$, the resulting problem $\max\{A^T u = P^T \lambda, u \geq 0\}$ is the dual of the weighted sum linear programme with $w = \lambda$. The duality for linear programming ensures that, assuming that $\mathcal{X} \neq \emptyset$ and $\lambda^T Px$ is bounded below over $\mathcal{X}$, the minimal value of the weighted sum linear programme with $w = \lambda$ is equal to the maximal value of its dual. Combining this with the fundamental theorem of Isermann, it is possible to show that y^* is a weakly non-dominated point of $P(\mathcal{X})$, if and only if there exist $\lambda^* \geq 0 : e^T \lambda^* = 1$ such that $\max\{b^T u : A^T u = P^T \lambda^*, u \geq 0\}$ has an optimal solution u^* such that $b^T u^* = \lambda^* y^*$. $D(u^*, \lambda^*)$ is then a $\mathcal{K}$-maximal point of $D(\mathcal{U})$. Moreover, for any $(u, \lambda) \in \mathcal{U}$ defining a $\mathcal{K}$-maximal point of $D(\mathcal{U})$, u is an optimal solution of the dual of the weighted sum problem and hence defines a weakly non-dominated point of $P(\mathcal{X})$ via linear programming duality.
For explaining more in details the geometric duality by Heyde and Löhne, it is necessary to introduce the extended polyhedral image sets $\mathcal{P} = P(\mathcal{X}) + \mathbb{R}^p_{\geq}$ and $\mathcal{D} = D(\mathcal{U}) - \mathbb{K}$, also called respectively upper image and lower image. It is known that the $\mathbb{R}^p_{\geq}$-minimal non-dominated points of $\mathcal{P}$ and $P(\mathcal{X})$ as well as the $\mathcal{K}$-maximal elements of $\mathcal{D}$ and $D(\mathcal{U})$ coincide. The geometric duality theory establishes a relationship between the weakly non-dominated vertices of $\mathcal{P}$ and the $\mathcal{K}$-maximal facets of $\mathcal{D}$ and between the weakly non-dominated facets of $\mathcal{P}$ and the $\mathcal{K}$-maximal vertices of $\mathcal{D}$. This is an extension of the well-known duality of polytopes to $\mathcal{P}$ and $\mathcal{D}$.

Definition 38 (Duality between polytopes)**.** *Two polytopes $\mathcal{G}$ and $\mathcal{G}^*$ in $\mathbb{R}_p$ are said to be dual each other if there exists a one-to-one mapping Ψ between the set of all faces of $\mathcal{G}$ and the set of all faces of $\mathcal{G}^*$ such that Ψ is inclusion-reversing.*

The geometric duality theorem states that there is a similar duality relationship between $\mathcal{P}$ and $\mathcal{D}$. To better explain, the following notation is introduced.
Let $\varphi : \mathbb{R}_p \times \mathbb{R}_p \to \mathbb{R}$ be a coupling function defined by:

$$\varphi(y, v) = \sum_{i=1}^{p-1} y_i v_i + y_p(1 - \sum_{i=1}^{p-1} v_i) - v_p$$

If the values of the primal and dual objective functions are chosen as arguments, for $x \in X$ and $(u, \lambda) \in \mathcal{U}$, the following relationship is obtained:

$$\varphi(Px, D(u, \lambda)) = \lambda^T Px - b^T u$$

that is the difference between the value of the weighted sum linear programming problem with $w = \lambda$ and the value of its dual at u. Using this function, the following two set-valued maps are defined.

$$H : \mathbb{R}_p \to \mathbb{R}_p, \quad H(v) = \{y \in \mathbb{R}_p : \varphi(y, v) = 0\}$$
$$H : \mathbb{R}_p \to \mathbb{R}_p, \quad H^*(v) = \{v \in \mathbb{R}_p : \varphi(y, v) = 0\}$$

$H(v)$ and $H^*(y)$ are hyperplanes in $\mathbb{R}_p$ for all $v, y \in \mathbb{R}_p$. By means of the definitions.

$$\lambda(v) := (v_1, ..., v_{p-1}, 1 - \sum_{i=1}^{p-1} v_i)^T$$
$$\lambda^*(y) := (y_1 - y_p, ..., y_{p.1} - y_p, -1)^T,$$

they can be expressed as:

$$H(v) = \{y \in \mathbb{R}_p : \lambda(v)^T y = v_p\},$$
$$H^*(y) = \{v \in \mathbb{R}_p : \lambda^*(y)^T v = -y_p\}.$$

The map H can be used to define the duality map $\Psi : 2^{\mathbb{R}_p} \to 2^{\mathbb{R}_p}$. Let $\mathcal{F}^* \subseteq \mathbb{R}_p$, then

$$\Psi(\mathcal{F}^*) = \bigcap_{v \in \mathcal{F}^*} H(v) \cap \mathcal{P}$$

Theorem 16 (Geometric duality theorem)**.** *Ψ is an inclusion reversing one-to-one map between the set of all proper $\mathcal{K}$-maximal faces of $\mathcal{D}$ and the set of all proper weakly non-dominated faces of $\mathcal{P}$ and the inverse map is given by*

$$\Psi^{-1}(\mathcal{F}) = \bigcap_{y \in \mathcal{F}} H^*(y) \cap \mathcal{D}$$

Moreover for every proper $\mathcal{K}$-maximal face $\mathcal{F}^*$ of $\mathcal{D}$ it holds $dim\mathcal{F}^* + dim\Psi(\mathcal{F}^*) = p - 1$.

In the proof of this important result, for which the reader can refer to [50], the following two pairs of dual linear programming problems play an important role:

$$\min_{x \in \mathcal{X}} \lambda(v)^T Px \quad \mathcal{X} = \{x \in \mathbb{R}^n : Ax \geq b\}$$

$$\max_{u\in\mathcal{T}(v)} b^T u \quad \mathcal{T}(v) = \{u \in \mathbb{R}^m : u \geq 0,\ A^T u = P^T \lambda(v)\}$$

and

$$\min_{x\in\mathcal{S}(y)} z \quad \mathcal{S}(y) = \{(x, z) \in \mathbb{R}^n \times \mathbb{R} : Ax \geq b, Px - ez \leq y\}$$

$$\max_{(u,\lambda)\in\mathcal{U}} (b^T u - y^T \lambda) \quad \mathcal{U} = \{(u, \lambda) \in \mathbb{R}^m \times \mathbb{R}^p : (u, \lambda) \geq 0, A^T u = P^T \lambda, e^T \lambda = 1\}$$

Two important consequences of the geometric duality theorem are represented by the following two corollaries that put in relation $\mathcal{K}$-maximal vertices of $\mathcal{D}$ and weakly non-dominated facets of $\mathcal{P}$ respectively weakly non-dominated vertices of $\mathcal{P}$ with $\mathcal{K}$-maximal facets of $\mathcal{D}$.

Corollary 2. *The following statements are equivalent*
(i) v is a $\mathcal{K}$-maximal vertex of $\mathcal{D}$,
(ii) $H(v) \cap \mathcal{P}$ is a weakly non-dominated facet of $\mathcal{P}$.
Moreover, if $\mathcal{F}$ is a weakly non-dominated $(p-1)$-dimensional facet of $\mathcal{P}$, there is some uniquely defined ppint $v \in \mathbb{R}_p$ such that $\mathcal{F} = H(v) \cap \mathcal{P}$.

Corollary 3. *The following statements are equivalent*
(i) y is is a weakly non-dominated vertex of $\mathcal{P}$,
(ii) $H^*(y) \cap \mathcal{D}$ is $\mathcal{K}$-maximal facet of $\mathcal{D}$.
Moreover, if $\mathcal{F}^*$ is a $\mathcal{K}$-maximal $(p-1)$-dimensional faced of $\mathcal{D}$, there is some uniquely defined point $y \in \mathbb{R}_p$ such that $\mathcal{F}^* = H^*(y) \cap \mathcal{D}$.

3.2 Multiobjective integer linear programming

A multi-objective integer linear programming problem can be formulated as follows:

$$\min Cx$$

$$Ax = b$$

$$x \geq 0 \quad (integer)$$

As in the continuous case, $x \in \mathbb{Z}^n$ is the vector of decisional variables, $C \in \mathbb{R}^{p\times n}$ is the coefficient matrix defining the p objectives, $A \in \mathbb{R}^{m\times n}$ is the coefficient matrix and $b \in \mathbb{R}^m$ is the vector defining the right hand side of the problem constraints. Differently from multiple objective linear programming, because of the presence of integer constraints on the decisional variables, it is not sufficient to aggregate the objectives through weighted sums to generate the set of all efficient solutions. This means that in the discrete case, 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 $X_{NE} = X_E \backslash X_{SE}$. Their images in the objective space are denoted respectively by Y_{NN} and Y_{SN} (see Figure 3.3 for a generic representation in the bi-objective case). Usually there are many more non-supported than supported efficient solutions (see [79]) and they significantly contribute to the difficulty of the multiple objective integer linear programming

problems. Indeed, these problems are in general $\mathbb{NP}$-hard even for those that have efficient algorithms in the single objective case. Moreover the number of efficient solutions (and of non-dominated points) for such problems may be exponential in the problem size. This prohibits the development of any efficient method to determine all efficient solutions. Even the size of the set of supported efficient solutions may be exponential, however numerical results show the number of supported efficient solutions grows linearly with the problem size but the number of non-supported efficient solutions grows as an exponential function (see [79]). Among the solution approaches for multiple objective integer linear programming problems it is possible to distinguish between exact and approximation methods. For the aim of this work, only the first class of methods will be considered in more details, focusing on the scalarization techniques in Section 3.2.1, while in the last section only a glimpse will be given about approximation methods.

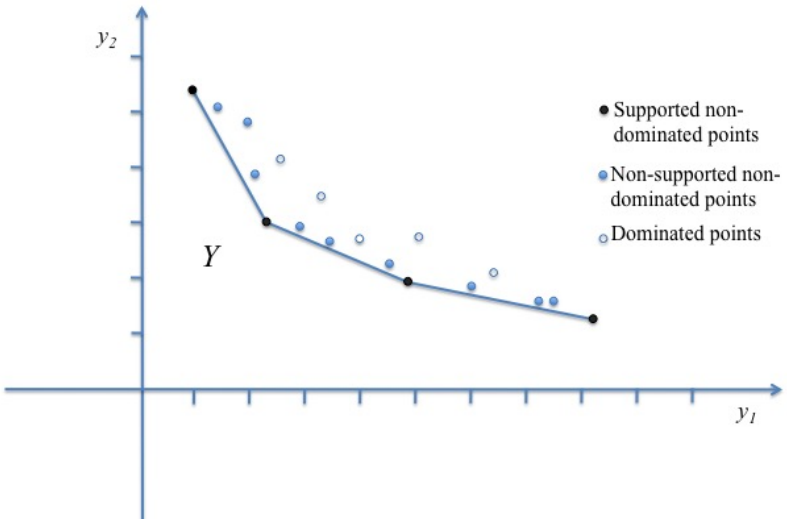


Figure 3.3. Example of Objective Space for a Bi-Objective Integer Linear Programme

3.2.1 Scalarization techniques

In this section the main exact solution methods for multi-objective integer linear programming, based on scalarization, are summarized. A scalarization is a single objective problem related to the original multiple objective problem with additional variables and/or parameters, that is usually solved iteratively in order to find some subset of efficient solutions of the multi-criteria problem under consideration. Independently of the continuous or discrete nature of the problem, there are some desiderable properties of scalarizations.

- Correctness: this property assures that the optimal solutions of the single objective problem are (weakly) efficient for the original multiple objectives problem.

- Completeness: this property guarantees that, solving iteratively the single objective problem, all efficient solutions can be found.

- Computability: the scalarization should be not harder than single objective version of the problem (from theoretical and practical point of view).

- Linearity: scalarization has linear formulation.

In the following, some of the most popular scalarizations methods are presented, reporting for each of them which of these properties are satisfied.

The weighted sum method

Many of the solution methods for multiple objective integer linear programming combine the multiple objectives into one single objective. The most popular and the one used first is, as for the continuous case, the weighted sum scalarization. Then the problem solved is:

$$\min \lambda^T C x$$
$$Ax = b$$
$$x \geq 0 \quad (integer)$$

where $\lambda \in \mathbb{R}^p$ is such that $0 < \lambda_j < 1 \quad \forall j = 1, ..., p$ and $\sum_{j=1}^{p} = 1$. Varying the weights it is possible to generate all supported efficient solutions (see [41]). The most important advantage of this method is that for each $\lambda \in \mathbb{R}^p$ the problem is only as difficult as its single objective version.

The ϵ-constrained method

Another very popular method, able to find all efficient solutions, is the ϵ-constrained method (see [18]), where one of the p objectives (say j-th) is retained for minimization and the other $p-1$ are turned into constraints:

$$\min C_j x$$
$$Ax = b$$
$$C_k x \leq \epsilon_k \quad k \neq j$$
$$x \geq 0 \quad (integer)$$

All efficient solutions can be found by appropriately specifying the right hand side values ϵ_k (see [32]). The disadvantage of this method is the presence of the upper bound constraints on the objective values, that are knapsack constraints and this makes the problem usually $\mathbb{NP}$-hard. In these cases, the scalarized problem is harder than the single objective version. The computational effort of this method is then strongly dependent on the problem under consideration.

Compromise method

Another well known approach is the compromise solution method, where the distance to an ideal point z^I or to an utopian point $z^U = z^I - \epsilon e$ is minimized, with $e = \{1, ..., 1\} \in \mathbb{R}^p$ and $\epsilon > 0$. The ideal point is defined according with the individual minimum value of each objective:

$$y_j^I = \min y_j(x) = \min C_j x$$
$$Ax = b$$
$$x \geq 0 \quad (integer)$$

Usually the most used distance measure is the Chebychev norm and the problem becomes:

$$\min \max_{j=1}^{p} \lambda_j \mid y_j(x) - y_j^I \mid$$
$$Ax = b$$
$$x \geq 0 \quad (integer)$$

From a theoretical point of view this method is able to produce the whole set of efficient solutions (see [32]), but when sum objectives are considered, this kind of problem is usually $\mathbb{NP}$-complete and for this reason it is rarely used.

Ranking methods

One approach that is successfully used for bi-criteria problems, is represented by ranking methods. The Nadir point is defined as follows:

$$y_j^N = \min\{y_j(x) : y_i(x) = y_i^I, \quad j = 1, 2 : \ i \neq j\}$$
$$Ax = b$$
$$x \geq 0 \quad (integer)$$

The ideal point $y^I = (y_1^I, y_2^I)$ and Nadir point $y^N = (y_1^N, y_2^N)$ define lower and upper bounds on the objective values of efficient solutions. Then starting from a solution with $y_1(x) = y_1^I$, and finding second best, third best and so on with respect to the first objective until y_1^N is reached, it is possible to generate the efficient set (see for example [29]).

The elastic constraint method

As previously mentioned, the disadvantage of the weighted sum method is the inability to generate all efficient solutions, while the problem of the ϵ-constrained method is strictly dependent on the upper bounds on the objective functions, that are necessary to find non-supported solutions. From these two observations it derives the idea of the elastic constraint method proposed by Ehrgott (see [32]). In this scalarization technique the ϵ-constraints are made elastic in order to obtain a problem easier to solve. Indeed it is allowed to violate the upper bounds on the

objective values, introducing a penalty associated with the constraints violation. The scalarized problem is the following:

$$\min C_j x + \sum_{k \neq j} \mu_k s_k$$
$$Ax = b$$
$$C_k x + l_k - s_k = \epsilon_k \quad k \neq j$$
$$s_k, l_k \geq 0 \quad k \neq j$$
$$x \geq 0 \quad (integer)$$

The parameters are the penalty coefficients μ_k and the right hand side values ϵ_k. Two sets of additional variables are introduced, slack variables l_k and surplus variables s_k to turn the upper bound on the objective values into equality constraints. This method is able to generate all efficient solutions (see [32]) and, if applied properly, reduces the computational effort required to solve the scalarized problem to acceptable levels. Indeed, the method of elastic constraints allows as much as possible the utilization of the structure of the single objective problem, which the ϵ-constrained method destroys. Moreover such a method contains as special cases the weighted sum method and the ϵ-constrained method.

3.2.2 Other methods

Among the exact approaches for the resolution of multi-objective integer linear programming problems, it is possible to mention the adaptation of the branch&bound technique to the multiple objectives case (see for example [45] where an adaptation of the branch&bound is used to deal with the multi-objective lot sizing problem). It is well known that such a procedure consists in partitioning the problem into mutually disjoint subproblems. For each subproblem a bound is computed and the process continues until an optimal solution is found. The difficulty in the application of this method to multi criteria integer problems is due to the computation of the bounds for the subproblems, that are Nadir points. Indeed, these may be difficult to compute or not effective in discarding a significant number of feasible non-efficient solutions. For such a reason only few papers examined this solution method.
Another possible exact approach it is represented by the use of single objective methods for a particular problem adapted to the multi-objective case (see for example [26] and [48] for adaptations of Prim's algorithm to its multi-objective version or [30] and [14] for the out-of-kilter and the network simplex method to multiple objectives network flow problem).
Eventually, a general framework for the exact solution of multi-objective integer linear programming problems is the two-phase method. This approach computes the whole set of efficient solutions in two steps. In the first phase the supported efficient solutions are found, usually using the scalarization techniques and solving single objective problems. In the second phase the non-supported efficient solutions are generated using specific methods according with the problem and the bounds. Such approach will be illustrated more in details in the next chapter with particular reference to the bi-objective min cost flow problem and to the bi-objective minimum spanning tree problem.

Approximation methods

Before concluding this chapter, in order to give a more complete idea about the solution approaches to multiple objective integer linear programming, a very short introduction to the approximation methods is presented.
It is well known that in the single objective case a valid alternative to exact methods for solving large-scale instances of integer linear programming problems is to design an approximation method. This is true also in the multi-objective case for which such methods may represent a good tradeoff between the quality of an approximation of the efficient solutions set and the time and memory required. Adaptations from the single to the multiple objective case have concerned genetic algorithms, simulated annealing (see for example [75]), tabu search (see for example [27]), and the greedy randomized adaptive search procedure (see for example [40]). In this context the following two main approaches can be distinguished:

- *Methods of local search in objective space.* The procedure, starting from an initial solution, approximates a part of the non dominated frontier corresponding to a given search direction λ. An aggregation of the objectives, often based on the weighted sum, has the effect to focus the search on a part of the non-dominated frontier. Such strategy is repeated for several search directions, in order to approximate the whole non-dominated frontier. According to the methods, the directions can be defined a priori, interactively, or aleatory. At each iteration, the search procedure uses only one solution and tries to attract the solution generated towards the set of efficient solutions along direction λ. The definition of λ plays an important role on the efficiency of these adaptations.

- *Population based methods.* Differently from the first approach, in which only one solution is used to start the procedure for approximating the non-dominated frontier, in this case all the population contributes to the evolution process toward the efficient solutions set. Considering a population of solutions, these methods can search for many efficient solutions in parallel by means of self adaptation and cooperations. This feature makes the population-base procedures attractive for solving multiple objective integer linear programming problems.

Chapter 4

A new two-phase strategy for solving bi-objective integer network flow problems

This chapter presents a new two-phase solution approach for the bi-objective integer network flow problem and its specializations. This solution method was introduced for the first time by Ulungu and Teghem in [78] and consists in generating a complete set of efficient solutions by means of two steps. The first one provides the supported efficient solutions, the second one generates a complete set of non-supported efficient solutions. Indeed, the class of problems under consideration, because of the presence of integer variables, is characterized by the existence of non-dominated points that lie in the interior of the convex hull of Y called non-supported non-dominated points. As explained in Chapter 3, several techniques could be used for the generation of supported Pareto optimal solutions. On the contrary, as regards those non-supported, it does not exist an efficient general procedure for their generation because of the lack of a theoretical characterization of these solutions at the current state of the art. The prevalent strategy in the bi-objective case, used in the second phase, for generating them, is the restriction of the exploration area in the outcome set, to the triangles that can be built considering consecutive pairs of supported non-dominated points sorted in increasing order according with the first objective. In each of these triangles, the exploration for finding non-supported non-dominated points, depends on the specific problem under consideration. This work proposes a general procedure for implementing this part. This procedure is applicable to a generic bi-objective integer network flow problem with appropriate specializations according with the problem under consideration. In Section 4.2 and in Section 4.3 a detailed description of this procedure will be given.

4.1 Two-phase method for bi-objective combinatorial optimization problems

As previously mentioned, for combinatorial optimization problems with two or more objectives, there exist non-dominated points that lie in the interior of the convex

hull of Y. In this context one of the possible strategies for finding a complete set of Pareto optimal solutions is represented by the so called two-phase approach. This is a general framework for the exact solution of multi-criteria combinatorial optimization problems. It consists in generating a complete set of efficient solutions in two steps: in the first one the supported efficient solutions are generated, 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. According with the specific problem that has to be solved, several standard techniques can be used in the first phase (for example the weighted sum method, the ϵ-constrained, or the lexicographic method, see [32] and [33]). As regards the second phase, focusing on the bi-objective case, the area to be explored for generating the remaining non-supported solutions is limited using consecutive pairs of supported points, sorted in increasing order with respect to the first objective. The deletion of the points dominated by the pair of supported points under consideration leads to triangles whose vertices are the two supported points and the corresponding Local Nadir Point. The area that has to be explored for finding non-supported non-dominated points is therefore limited to these triangles. In each of them, new feasible points for the bi-objective problem are generated. Among these solutions the non-dominated are identified. The procedure applied for this generation depends on the specific problem under consideration. In [70] a k-best flow algorithm is used for generating feasible flows for a single objective min cost flow problem, ordered according to their objective value. At each iteration, starting from the initial optimal flow, the incremental graph is built. On this graph the minimum cost cycle is found and used for defining the second best flow. At this point two new networks are identified: one in which the original optimal flow is infeasible and the second best is optimal and one in which the original optimal flow is still optimal and the second best flow is infeasible. On each of these two new networks the same procedure as before is applied. Two new feasible flows are generated. The minimum among these two will be the third best flow and so on. In [77] a k-best algorithm is applied for producing 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 $e \in T$, $f \in E - T$ and e, f has the smallest weight possible, is applied. In this way the second best spanning tree is generated. At the generic iteration j, for each of the spanning trees generated until the iteration $j-1$ (T_1,...,T_{j-1}), two lists of arcs are created. The first one (IN) is the list containing the arcs in T that must remain in T and the second one (OUT) is the list of arcs that are out of T and that must remain out of T. At each iteration a T-exchange e, f such that $e \in T - IN$, $f \in E - T - OUT$ and e, f has the smallest weight possible, among all the possible T-exchanges applicable on the previous spanning trees generated until the current iteration, is applied. A new spanning tree is so generated, the one with the minimum possible deterioration of the objective value.

In the works mentioned above, the procedures applied generate in order the feasible solutions, but their operating principle is different and strictly related to the specific problem for which they were designed. In the following section a general procedure is proposed, able to generate all the feasible solutions for an integer network flow problem. This strategy is based on the analysis of the reduced costs associated with the arcs of the network and, by means of appropriate specializations, will be

applied to the bi-objective integer minimum cost flow problem and to the bi-objective minimum spanning tree problem respectively in Chapter 5 and in Chapter 6.

4.2 A recursive procedure for generating all the feasible solutions of a single objective integer network flow problem

In this section a new general recursive procedure for generating feasible solutions for a single objective integer network flow problem is described. This procedure is based on the analysis of the reduced costs of the network arcs. More in details, given a network flow problem $P(G = (N, A))$ with integer flow variables, let x^* be an optimal basic feasible solution for such a problem. It is well-known that, in correspondence to an optimal basic feasible solution, a reduced cost is associated to each arc (variable). The reduced cost of a variable represents the increase of the objective value if the value of the variable is increased (or decreased) by one unit (see [12]).
Starting from this observation, the idea of the recursive procedure presented in this work is to consider the non-zero reduced costs (assuming that x^* is not degenerate) associated to the decision variables of the flow problem, sorting them from the smallest to the biggest in absolute value. Starting from the smallest one, the corresponding variable is modified (increased/decreased by one unit). In order to obtain a new feasible solution by means of this change, it is also necessary to identify the fundamental cycle that the arc creates with the arcs in the solution tree. On the arcs in the fundamental cycle, appropriate changes have to be made for having a new solution satisfying all the problem constraints. These changes depend on the specific problem and they will be explained in detail in Chapters 5 and in Chapter 6 for two fundamental network flow problems: the min cost integer flow and the minimum spanning tree problems.
When the arc under consideration is the one with the smallest non-zero reduced cost in absolute value, the new feasible solution obtained applying this procedure will correspond to the second best solution. Indeed, the objective value for the new solution will be equal to the previous one plus the smallest non-zero reduced cost in absolute value. This means that the increment of the objective function will be the smallest possible one, that is the new feasible solution will be the second best one. After the generation of the second best solution, the changes, mentioned above, on the same arc (the one with smallest reduced cost in absolute value) and on the arcs of the fundamental cycle that it creates, are made again until all the problem constraints continue to be satisfied, in order to generate the third best solution, the fourth best solution and so on. More precisely, for the min cost integer flow problem, the number of feasible solutions that can be generated in strict ranking order considering the smallest non-zero reduced cost in absolute value, is equal to the minimum difference between the original (optimal) flow and the lower bound (or the upper bound and the original (optimal) flow) among the arc corresponding to this reduced cost and all the arcs in the fundamental cycle that it creates with the basis at the optimal basic solution. Considering the minimum spanning tree problem, this number will be equal to the number of arcs, in the fundamental cycle,

that can be exchanged with the one under consideration.
After the analysis of the smallest non-zero reduced cost in absolute value, the feasible solutions generated by this procedure will not be in strict ranking order. Indeed, analyzing the successive non-zero reduced costs, in order not to miss feasible solutions, it is necessary to consider also combinations of arcs associated with different non-zero reduced costs. More in details, when a new arc (associated with a given non-zero reduced cost) is examined, a new feasible solution is generated with the same strategy described above for the second best solution. After that, before repeating the changes on the same arc, the non-zero reduced costs already analyzed are again examined in the same order (from the smallest to the current one) to obtain, if it is possible, new feasible solutions from the current one. For such a reason, this strategy is designed as a recursive procedure. A pseudocode of this algorithm is listed below.

INPUT: graph $G = (N, A)$, optimal solution x^*, set of the problem constraints X, vector of non-zero reduced costs in absolute value $\bar{c}^* = \{c_1^*, c_2^* ... c_K^*\}$ (sorted from the smallest to the biggest) where K is the number of non-zero reduced costs, and the list $FS = \{x^*\}$ initialized with the optimal basic feasible solution x^*.

find_feasible_solutions($G = (N, A)$, x^*,X, $\bar{c}^*$, FS)
1: $it = 1$
2: **while** $it \leq K$ **do**
3: Let $x^*_{(i,j)}$ be the flow variable in the original optimal basic feasible solution x^* corresponding to the element $c^*_{it} \in \bar{c}^*$ (c_1^* is the minimum non-zero reduced cost in absolute value).
4: Let C^* be the cycle that the arc (i, j) creates with the arcs belonging to the spanning tree corresponding to the optimal basic feasible solution x^*.
5: Let $M(C^*)$ be the set of changes to be applied on the fundamental cycle C^* to obtain a new solution.
6: **if** $M(C^*)$ on x^* satisfy X **then**
7: x^{it}=change_flow((i, j), C^*)
8: Insert the new feasible solution x^{it} so obtained in FS.
9: **find_feasible_solutions**($G = (N, A)$, x^{it}, X, $\bar{c}^*_{it}$, FS)
10: **end if**
11: $it = it + 1$
12: **end while**

OUTPUT: Set FS containing all feasible solutions.

Algorithm 1: general recursive algorithm.

The input of this recursive procedure is: the network, the problem constraints, the optimal basic feasible solution, and the vector of non-zero reduced costs sorted from the smallest to the biggest in absolute value. The procedure stops when all the non-zero reduced costs are analyzed. The feasible solutions found by means of this algorithm are not in strict ranking order, with the exception, as mentioned above, of those generated considering the smallest reduced cost in absolute value. However, 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 smallest to the biggest one in absolute value, at each branch of the tree certainly the value of the objective function will get worse. From this observation it is possible to adapt this procedure in the context of a two-phase method for the generation of non-supported efficient solutions as described in the following section.

4.3 Putting things together

At this point, after the description of the general recursive procedure for generating feasible solutions for an integer network flow problem, it is possible to introduce an adaptation of this algorithm for finding non-supported non-dominated points of a bi-objective integer network flow problem. In this case, as mentioned in Section 4.1 a possible strategy for finding a complete set of efficient solutions is represented by the two-phase strategy. Focusing on the second phase, the one by means of which the non-supported Pareto optimal solutions are identified, it is possible to use this general recursive procedure for exploring the areas of the triangles obtained, as explained previously, by means of the supported non-dominated points associated with the supported efficient solutions given by the first phase. Indeed, even if the general procedure presented before does not generate the feasible solutions in a strict ranking order, at each branch, it is sure that the new solution found will be equivalent or worse than the one from which is generated. This observation permits to apply this strategy in each triangle, limiting the generation of feasible points by means of an appropriate upper bound without missing any feasible non-dominated point. The single objective problem is obtained from the original one defining the weighted sum of the two objectives. The weights are computed as the differences between the coordinates, in the objective space, of the two points corresponding to the two supported efficient solutions from which the triangle is built (see [69]). At each iteration, once a new feasible solution is determined, a check on the corresponding point in the outcome set is made. If the point associated with the new solution is non-dominated by points already generated, and belongs to the triangle under consideration, the corresponding solution is inserted in the set of efficient solutions. In the recursive procedure proposed in this work the same check, after the generation of a new feasible solution, is made. Differently from the previous works, as explained above, this procedure does not generate the feasible solutions in strict ranking order, and this check has to be repeated when the exploration of each triangle terminates. Indeed, during the recursive procedure, a feasible solution, that dominates another one previously generated, could be produced. For this reason, at the end of the recursive procedure applied in a given triangle, it is necessary to repeat the dominance check among the points in the objective space associated with the feasible solutions generated, in order to eliminate potential dominated points.
After the dominance check, if the point belongs to the current triangle, it is used to update the upper bound for limiting the generation of new feasible points. This update is made with the same strategy explained in [69]. The point corresponding to a new feasible non-dominated solution, together with the non-dominated points already generated in the same triangle, are sorted in increasing order with respect to the first objective. Consecutive pairs of these points (included the supported points from which the triangle is built) are considered and the line joining such pair, parallel to the one joining the two supported points associated with the triangle under consideration, is built. Among these lines, the new upper bound will be defined equal to the right hand side of the equation associated with the line that has the maximum distance from the line joining the two supported points. This strategy is repeated for each triangle and permits to obtain in the end a complete set of efficient solutions for the bi-objective integer network flow problem under consideration. A

pseudocode of the second phase adopting such a procedure is showed below.

INPUT: Network $G = (N, A))$, costs matrix $c = (c^1, c^2)$, list of extreme efficient solutions $x^1, ..., x^s$.

Two_phase_bflow

1: $i = 1$
2: $\mathcal{E} = \emptyset$
3: **while** $(i < s)$ **do**
4: $\mathcal{E}_i = \{x^i, x^{i+1}\}$
5: Compute $\lambda_1 = y_2(x^i) - y_2(x^{i+1})$, $\lambda_2 = y_1(x^{i+1}) - y_1(x^i)$ and $c_\lambda = \lambda_1 c^1 + \lambda_2 c^2$
6: **if** $\mathcal{E} \neq \emptyset$ **then**
7: **for** $x^b \in \mathcal{E}$ **do**
8: **if** $y(x^b) \in T_i$, it is not dominated and not equivalent to any $x \in \mathcal{E}_i$ **then**
9: Insert x^b in $\mathcal{E}_i$ and remove x^b from $\mathcal{E}$
10: **end if**
11: **end for**
12: $\Delta = \max\{\lambda_1(y_1(x^{i,j+1}) - 1) + \lambda_2(y_2(x^{i,j}) - 1), j = 0, ..., r\}$, $r + 1 =\mid \mathcal{E}_i \mid$
13: **end if**
14: **else**
15: $\Delta = \mu_\lambda = \lambda_1(y_1(x^{i+1}) - 1) + \lambda_2(y_2(x^i) - 1)$ /* initial value of Delta*/
16: $\mathcal{E} =$ **find_feasible_solutions**$[G(N, A), x^{i+1}, X, \bar{c}^{i+1}, \mathcal{E}, \Delta]$ /*with $\bar{c}^{i+1}$ vector of non-zero reduced costs in absolute value associated with x^{i+1} (sorted from the smallest to the biggest) and X set of problem constraints*/
17: **for** $x^b \in \mathcal{E}$ **do**
18: **if** $y(x^b) \in T_i$, it is not dominated and it is not equivalent to any $x \in \mathcal{E}_i$ **then**
19: Insert x^b in $\mathcal{E}_i$ and remove x^b from $\mathcal{E}$
20: **end if**
21: **else**
22: Insert x^b in $\mathcal{E}$
23: **end for**
24: $i = i + 1$
25: **end while**

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

Algorithm 2: Two-phase algorithm.

Chapter 5

A new algorithm for the bi-objective integer min cost flow problem

In this chapter we present a new two-phase algorithm for finding a complete set of efficient solutions for the bi-objective integer minimum cost flow problem described in [6]. As mentioned in Chapter 3, one of the possible strategies to approach the multi-objective integer linear programming problems consists in generating Pareto optimal solutions in two phases (see [78]). In the first phase, only the extreme non-dominated points, that is vertices of the convex hull of Y, are found. In the second phase, by means of an exploration in the outcome set, limited by the points associated with the extreme efficient solutions generated in the first phase, others non-extreme Pareto optimal, supported and non-supported, solutions are produced. According with the specific problem under consideration, different methods can be used for implementing the first phase. As regards the computation of the remaining efficient solutions in the second phase, an enumerative approach has to be used, as at current state of the art, a theoretical characterization of these solutions does not exist.

5.1 State of the art

Most of the works on the exact solution of the bi-objective integer minimum cost flow problem, consist in two-phase methods (see [47] for a complete review on multi-objective minimum cost flow problems). As mentioned in Chapter 4, in the first phase any approach to solve the continuous version of the problem can be used for finding a complete set of extreme supported efficient solutions. Then the focus of researchers was mainly on the possible implementations of the second phase. To this end in [62], Lee and Pulat investigated the structure of the solutions of the bi-objective integer min cost flow problem and performed an implicit search of the decision space. More in details, they considered adjacent basic solutions (corresponding to extreme supported non-dominated points) to generate intermediate solutions by means of T-exchanges and modifying lower or upper bounds of the arcs. After all the candidates have been generated, a filtering process is applied to delete dominated

points from the candidate set. According to the authors, this procedure is able to generate all non-supported efficient solutions, assuming non-degeneracy. In [51] an extension of this algorithm was proposed for dealing also with degenerate instances by Huarng, Pulat and Ravindran. In [73], Sedeño-Noda and González-Martín showed that this algorithm is incorrect, because introducing only two non-basic arcs at a time, like proposed by Lee and Pulat, it is possible to miss some efficient flows. Moreover, they presented a left-right approach starting from an efficient solution that is optimal for the first objective and moving to adjacent solutions in order to find non-supported efficient flows. In this algorithm also combinations of non-basic variables are considered. In their numerical tests they showed that the number of non-supported non-dominated points grows faster than the number of supported non-dominated points in relation to the size of the networks. In [68] an example of a network where there exists an efficient solution that is not adjacent to any of the others efficient solutions, is given to show the incorrectness of the approach proposed by González-Martín. In [37] Figueira proposed a branch&bound approach to find all non-dominated points, solving repeatedly ϵ-constrained problems in which the second objective function is added to the constraints. In [70] Ehrgott and Raith presented the most recent two-phase approach. After the generation of the supported extreme efficient solutions by means of a parametric network simplex, in the second phase an adaptation of the k-best algorithm by Hamacher (see [46]) is used to generate the remaining efficient solutions. This algorithm is applied in each triangle that can be identified considering consecutive pairs of supported extreme non-dominated points sorted according with the first objective, in increasing order, and the associated local Nadir point. One of the two efficient solutions, corresponding to the two extreme non-dominated points, is taken as initial optimal solution for a weighted sum problem appropriately defined. From this solution the k-best algorithm is applied to generate the second best flow, the third best flow and so on. At each iteration, the point associated with the flow generated is computed. A check on this point is made to verify if it is in the current triangle under consideration and if it is non-dominated. If both these properties are satisfied, the flow associated with this point is inserted in the list of the non-supported efficient solutions. The procedure stops when all the triangles have been explored.

5.2 The general idea of the algorithm

In the first phase, because of the total unimodularity of the coefficient matrix related to the flow conservation constraints (all parameters and variables are supposed integer), it is possible to use any methodology for solving the bi-objective continuous network flow problem. For this reason, in the method proposed in this work, the dual variant of Benson's algorithm (see [34] and [49]) is used for generating the extreme efficient solutions. This latter is a solution approach for generic multi-objective linear programming problems that will be described in more detail in the next section. In the second phase, the remaining supported and non-supported non-dominated points are generated through an exploration of the objective space. In more details, consecutive pairs of non-dominated extreme points, sorted in increasing order with respect to the first objective ($y1$), are considered and the exploration is limited

to the triangles that is possible to build using these pairs and the corresponding Local Nadir Point (see Figure 5.1). Given a triangle, the supported efficient solution corresponding to one of its two vertices, is assumed as an initial optimal solution for the single objective flow problem obtained taking a combination, that we will describe in Section 5.4.4, of the two objectives. Starting from this solution a recursive algorithm, capable to generate all feasible flows of the single objective problem, is applied, setting the upper bound on its objective function, for limiting the generation of feasible flows, as in the work by Ehrgott and Raith ([70]). First, all the equivalent optimal solutions of the single objective problem are generated, that is all the non-extreme efficient solutions for the original bi-objective problem. Then, by means of the corresponding points, sorted in increasing order with respect to the first objective, a further reduction of the area to be explored in the triangle is made.

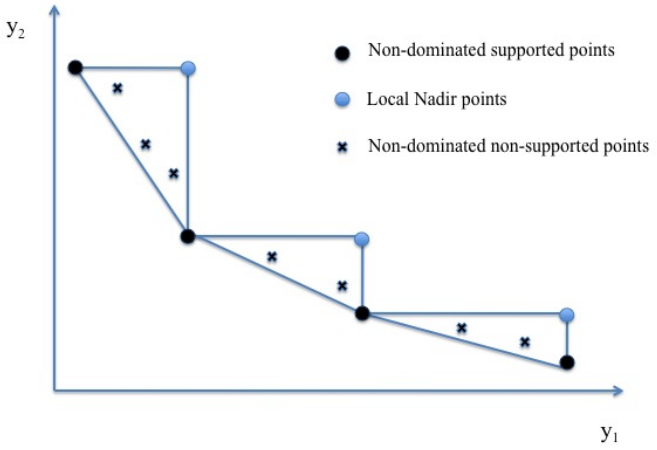


Figure 5.1. Example of triangles in the objective space

Indeed, it is possible to limit the search for non-supported non-dominated points only to the new smaller triangles, contained in the initial one, determined by considering consecutive pairs of these equivalent points and the corresponding Local Nadir Points (see Figure 5.2). Then, starting from each one of the solutions associated with these equivalent points, the recursive algorithm is applied for finding feasible flows corresponding to non-supported non-dominated points. At each call of the recursive algorithm, if the feasible flow produced corresponds to a "potentially non-dominated" point that belongs to the triangle under consideration, the upper bound in the current triangle is updated (see [69]). Then, the dual variant of Benson's algorithm is again applied to the network in which the initial equivalent optimal solution, for the single objective problem, is infeasible and the current feasible flow is optimal. A new set of efficient extreme points is so generated. Some of these points may lie in the current triangle, others are located out of the triangle. By means of the points

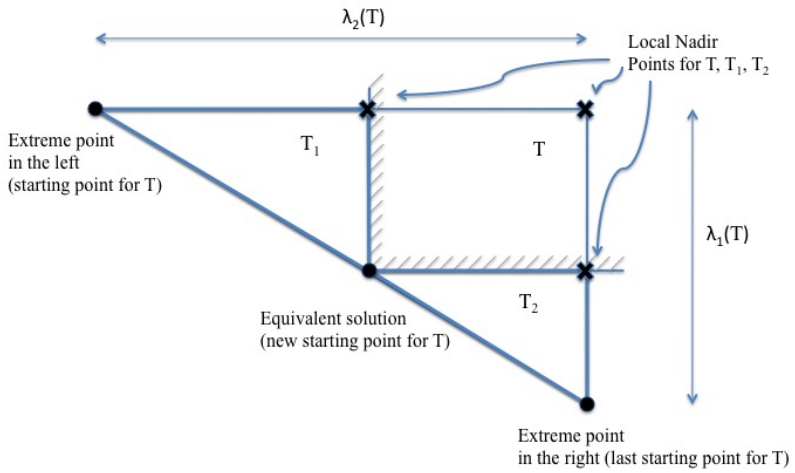


Figure 5.2. Example of Extreme and Local Nadir points of a given triangle

internal to the current triangle which are non-dominated, it is possible to further update the upper bound. The extreme points so generated which do not belong to the current triangle could lie in other triangles not yet explored. For this reason they are stored and used for possibly improving the upper bound in the next iterations of the algorithm. By reiterating this procedure in each triangle, it is possible to improve the upper bounds, further limiting the exploration area in the outcome set and speeding up the search for finding non-supported efficient solutions.

5.3 First phase: the dual variant of Benson's algorithm

In this section the solution method used in the first phase for finding the extreme efficient solutions is described. As mentioned above, the approach proposed for this part, is the dual variant of Benson's algorithm (see [34]), that is a method to solve multiple objective linear programs in the outcome space, constructing a sequence of approximating polytopes that contain the feasible set in the outcome space and terminates when all extreme points of the polytope are feasible for the original problem. Despite the full name, Benson's algorithm is an exact one. In [34] a dual variant of this approach is proposed, consisting in the application of Benson's algorithm (with some slight modifications), to the dual outcome set, using some results in the duality for multiple objective linear programs (see [50] and Section 3.1.2). This variant can be used in this context because it needs to solve linear programming problems that are single objective minimum cost flow problems, hence the total unimodularity property is preserved and this guarantees that the solutions will be integer.

5.3.1 Benson's algorithm strategy

In the primal algorithm the upper image $\mathcal{P}$ is built, while the dual variant of Benson's algorithm constructs the lower image $\mathcal{D}$ (see Chapter 2). From $\mathcal{D}$, using the geometric duality, $\mathcal{P}$ can be obtained. It is assumed that the primal feasible set $\mathcal{X}$ of the original multiple objective problem is nonempty and $\mathcal{P}$ is $\mathbb{R}^p_{\leqq}$-bounded from below. The dual variant of Benson's algorithm first builds a p-dimensional polyhedral set $\mathcal{S}^0 = \{v \in \mathbb{R}^p : \lambda(v) \geqq 0, \varphi(Px^0, v) \geqq 0\}$ such that $\mathcal{D} \subseteq \mathcal{S}^0$. As described in Chapter 3, $\lambda(v) \in \mathbb{R}^p$ is the weight vector whose components sum is equal to 1 and $\varphi(Px^0, v) : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}$ is the coupling function that defines the gap between the value of the weighted sum linear programming problem and the value of its dual. At each iteration a vertex s^k of $\mathcal{S}^{k-1}$ not contained in $\mathcal{D}$ is chosen and a supporting hyperplane to $\mathcal{D}$ is determined by solving the weighted sum linear program $(P1(v^k))$, where v^k is a boundary point of $\mathcal{D}$ on the line segment joining s^k with the interior point $\hat{d}$ of $\mathcal{D}$. The polytope $\mathcal{S}^k$ is defined by intersecting $\mathcal{S}^{k-1}$ with the halfspace of the hyperplane containing $\mathcal{D}$ until at termination $\mathcal{S}^{k-1} = \mathcal{D}$.

5.3.2 Main steps

Initialization $(k = 0)$.

(Step i1) Choose some $\hat{d} \in \text{int } \mathcal{D}$.
(Step i2) Compute an optimal solution x^0 of $(\text{P1}(\hat{d}))$.
(Step i3) Set $S^0 := \{v \in \mathbb{R}^p : \lambda(v) \geqq 0, \varphi(Px^0, v) \geqq 0\}$ and $k = 1$.

Iteration steps $(k \geq 1)$.

(Step k1) If vert $S^{k-1} \subseteq \mathcal{D}$ stop, otherwise choose a vertex s^k of S^{k-1} such that $s^k \in \mathcal{D}$.
(Step k2) Compute $\alpha^k \in (0, 1)$ such that $v^k := \alpha^k s^k + (1 - \alpha^k)\hat{d} \in \max_{\mathcal{K}} \mathcal{D}$.
(Step k3) Compute an optimal solution x^k of $(\text{P1}(v^k))$.
(Step k4) Set $S^k := S^{k-1} \cap \{v \in \mathbb{R}^p : \varphi(Px^k, v) \geqq 0\}$.
(Step k5) Set $k := k + 1$ and go to (Step k1).

5.4 Second phase: a recursive algorithm for generating all feasible flows

In the second phase, for generating the remaining supported and non-supported Pareto optimal solutions, an exploration in the outcome set is implemented. This exploration is limited, as mentioned in Section 5.2, to the triangles that can be obtained considering consecutive pairs of extreme non-dominated points provided by the first phase and their associated Local Nadir point. At each iteration of the algorithm, one of these triangle is analyzed, starting from the left, that is, from the triangle associated with the first two extreme non-dominated points sorted in increasing order with the respect of the first objective $(y1)$. Then, given the triangle under consideration at the generic iteration, the flow values associated with the

extreme efficient point on the right of the triangle, are used as initial optimal solution for the following single objective flow problem:

$$\min \sum_{(i,j)\in A} (\lambda_1 c^1_{(i,j)} x_{(i,j)} + \lambda_2 c^2_{(i,j)} x_{(i,j)}) \tag{5.1}$$

$$\sum_{j\in N^+(i)} x_{(i,j)} - \sum_{j\in N^-(i)} x_{(j,i)} = b_i \quad \forall i \in N \tag{5.2}$$

$$0 \le x_{(i,j)} \le u_{(i,j)} \quad \forall (i,j) \in A \tag{5.3}$$

$$x_{(i,j)} \quad \text{integer} \quad \forall (i,j) \in A \tag{5.4}$$

where N is the set of nodes and A is the set of arcs describing the network, u is the integer capacity vector, b is the integer demand vector, c^1 and c^2 are the integer costs vectors and λ_1 and λ_2 are the weights associated with each objective. These weights are defined as in [70]: given the two extreme efficient 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 defined as follows:

$$\lambda_1 = y_2(x^i) - y_2(x^{i+1}) \quad \text{and} \quad \lambda_2 = y_1(x^{i+1}) - y_1(x^i). \tag{5.5}$$

This definition of weights implies that the single objective flow problem has optimal solutions x^i and x^{i+1} and also all their convex combinations are optimal. Hence, it is possible to start from the solution x^{i+1} (or from the solution x^i) and consider it optimal for the weighted sum problem (5.1)-(5.4). Then, from this solution, a recursive algorithm, described next in detail, able to generate all the (integer) feasible flows of the problem is applied. In fact, all the other efficient solutions of the original bi-objective problem, correspond to flows that are feasible for the network under consideration and such that the associated points in the objective space are non-dominated. For this reason a new algorithm that provides all the feasible flows of a given network, starting from an initial optimal basic feasible solution, has been designed. This algorithm, differently from the k-best flow algorithm (see [46]), does not produce feasible flows ordered with the respect of the objective function, but according to its operating principle, at each iteration generates a feasible flow that is certainly worse, in term of objective value, than the previous one.

5.4.1 Strategy of the recursive algorithm

The procedure used for generating all the feasible flows for an integer min cost flow problem, with a single objective, works considering the reduced costs associated with the arc variables of the initial optimal basic feasible solution. The reduced costs are sorted from the smallest to the biggest one in absolute value and analyzed in this order. Given the smallest non-zero reduced cost in absolute value $r_{i,j}$, the fundamental cycle $C^*_{i,j}$ that the corresponding arc (i,j) creates with the spanning tree T^* associate with the starting optimal basic feasible solution x^*, is identified. From the arcs in the fundamental cycle, two distinct sets are generated: the set $CA_{i,j}$ of the arcs with the same orientation of the arc (i,j) and the set $DA_{i,j}$ of the arcs with opposite orientation to the arc (i,j). If it is true simultaneously that the arc (i,j) is empty, the arcs belonging to $CA_{i,j}$ are not saturated and the arcs in the

set $DA_{i,j}$ are not empty, then a new feasible flow is obtained increasing by one unit the flow on the arc (i, j) and on the arcs in $CA_{i,j}$ and decreasing by one unit the flow on the arcs in $DA_{i,j}$. Analogously, if it is true simultaneously that the arc (i, j) is saturated, that arcs belonging to $CA_{i,j}$ are not empty and that arcs in the set $DA_{i,j}$ are not saturated, a new feasible flow is obtained decreasing by one unit the flow on the arc (i, j) and on the arcs in $CA_{i,j}$ and increasing by one unit the flow on the arcs in $DA_{i,j}$.
At the first iteration of this algorithm, the feasible flow obtained is exactly the second best flow (see [3]), generated inserting in the basis associated with the initial optimal basic feasible solution, the arc with smallest (non-zero) absolute value of the reduced cost. Therefore, the corresponding deterioration of the objective function is as small as possible. At each iteration one of the bounds of the arc (i, j) is changed together with its flow: if the arc (i, j) is empty, its lower bound is increased by one unit, while if the arc (i, j) is saturated, its upper bound is decreased by one unit. Implementing this change, the new flow is an optimal basic feasible solution, for the new network, with the same basis as before. By means of this network change it is possible to reiterate the procedure starting from the new optimal basic feasible flow until we can change the flows (without violating constraints) on the arc (i, j) and on the arcs in the fundamental cycle that it creates with the spanning tree T^*. In doing so, we can continue this procedure for generating all the other worse feasible flows, as we can start again from an initial optimal basic feasible solution to which correspond the same reduced costs. When the flow value of one arc of the fundamental cycle reaches the value of its bound (lower or upper), the second non-zero reduced cost in absolute value has to be considered, starting again from the original network and the original optimal basic feasible solution. Considering the second non-zero reduced cost in absolute value $r_{i',j'}$, the same procedure described above is applied to generate a new feasible flow. In this case, before changing again the flow and the bound on the same arc $(i^{'}, j^{'})$, in order not to miss feasible solutions and to obtain the minimum possible deterioration of the objective function, starting from this new solution, the first non-zero reduced cost in absolute value has to be considered, until flow values of the arc (i, j) and of the arcs in the fundamental cycle created by (i, j) in the spanning tree T^* do not violate any constraint. Reiterating this procedure considering all the non-zero reduced costs in absolute value in order from the smallest to the biggest, and for all their possible combinations, as explained, all the feasible flows can be identified. Therefore, the resulting algorithm consists in a recursive procedure that is summarized in the following pseudocode.

INPUT: graph $G = (N, A)$, integer capacities vector k, optimal solution x^*, vector of non-zero reduced costs in absolute value $\bar{rc}^* = \{rc_1^*, rc_2^* ... rc_K^*\}$ (sorted from the smallest to the biggest) where K is the number of non-zero reduced costs, and the list $FS = \{x^*\}$ containing at the beginning the initial optimal solution.

find_feasible_flows($G = (N, A)$, u, x^*, $\bar{rc}^*$, FS)

1: $it = 1$
2: **while** $it \leq K$ **do**
3: Let $x^*_{(i,j)}$ be the flow variable in the original optimal basic feasible solution x^* corresponding to the element $c^*_{it} \in \bar{c}^*$ (c^*_1 is the minimum non-zero reduced cost in absolute value).
4: Let C^* be the cycle that the arc (i,j) creates with the arcs belonging to the spanning tree corresponding to the original optimal solution x^* in the original network (with capacities vector u).
5: **if** $((\bar{rc}^*_{it} > 0)$ && $(x^*_{(i',j')} < u_{(i',j')}\ \forall (i',j') \in C^* : (i,j)$ and (i',j') have same orientation) && $(x^*_{(i',j')} > 0\ \forall (i',j') \in C^* : (i,j)$ and (i',j') have opposite orientation)) **then**
6: Increase by one unit the flow and the lower bound on the arc (i,j) and the flow on the arcs $(i',j') \in C^*$ such that (i,j) and (i',j') have same orientation, decrease by one unit the flow on the arcs $(i',j') \in C^*$ such that (i,j) and (i',j') have opposite orientation.
7: Insert the new feasible flow x^{it} so obtained in FS.
8: **find_feasible_flows**($G = (N, A)$, u^{it}, x^{it}, $\bar{rc}^*_{it}$, FS).
9: **end if**
10: $it = it + 1$
11: **if** $((\bar{rc}^*_{it} < 0)$ &&$(x^*_{(i',j')} < u_{(i',j')}\ \forall (i',j') \in C^* : (i,j)$ and (i',j') have opposite orientation) && $(x^*_{(i',j')} > 0\ \forall (i',j') \in C^* : (i,j)$ and (i',j') have same orientation)) **then**
12: Decrease by one unit the flow and the upper bound on the arc (i,j) and the flow on the arcs $(i',j') \in C^*$ such that (i,j) and (i',j') have same orientation, increase by one unit the flow on the arcs $(i',j') \in C^*$ such that (i,j) and (i',j') have opposite orientation.
13: Insert the new feasible flow x^{it} so obtained in FS.
14: **find_feasible_flows**($G = (N, A)$, u^{it}, x^{it}, $\bar{rc}^*_{it}$, FS).
15: **end if**
16: $it = it + 1$.
17: **end while**

OUTPUT: Set FS containing all feasible flows.

Algorithm 3: recursive algorithm specialized for the integer min cost flow problem.

Theorem 17. *Let us assume that the integer min cost flow problem (5.1)-(5.4) has an unique optimal (basic) feasible solution x^* and let T^* be the spanning tree associated with x^*. Procedure* ***find_feasible_flows*** *examines all the feasible combinations of augmenting cycles generated by the $m - n + 1$ fundamental cycles associated with T^*.*

Proof. To prove this we just recall that Step 4 of the procedure **find_feasible_flows** examines first the fundamental cycle associated with the arc out of T^* with minimum non-zero reduced cost (in absolute value). Once all augmenting cycles associated with this fundamental cycle have been produced, it starts the generation of feasible combination of augmenting cycles in the following manner. The algorithm finds the fundamental cycle associated with the second non-zero reduced cost (in absolute value) and generates just one augmenting cycle associated with this fundamental cycle. Now, differently from the first non-zero reduced cost, the recursive procedure is called on the new feasible flow so obtained. This means that it attempts to combine the first augmenting cycle associated with the second non-zero reduced cost (in absolute value) with the first augmenting cycle associated with the first non-zero reduced cost (in absolute value). Note that continuing in generating augmenting cycles associated with the fundamental cycle related to the first non-zero reduced cost (in absolute value), all the augmenting cycles from the first non-zero reduced cost (in absolute value) are combined with the current augmenting cycle (in this case associated with the second non-zero reduced cost in absolute value). Then, when back on the feasible solution associated with the first augmenting cycle generated by the second fundamental cycle, the procedure seeks a further augmenting cycle

from the second fundamental cycle. If it succeeded, again the recursive procedure is called from the new feasible flow so generated. It should be clear that by means of this mechanism, all the feasible combinations of augmenting cycles generated by the first two fundamental cycles are examined. At the generic stage of the algorithm non-zero reduced cost k is considered and with the same mechanism all feasible combinations of augmenting cycles generated by the fundamental cycles associated with the first non-zero $k-1$ reduced costs (in absolute value) with the augmenting cycles associated with non-zero reduced cost k are generated. The algorithm continues generating feasible combinations of augmenting cycles in order of increasing reduced cost (in absolute value) until all non-zero reduced costs are examined, that is all the feasible combinations of augmenting cycles, including the whole set of fundamental cycles, are generated.

□

In order to prove a direct consequence of Theorem 17 we recall Theorem 1 from Chapter 1.

Theorem 1
Let $G = (V, E)$ be a connected graph and let $T = (V, E')$ be a spanning tree of G. Then the fundamental system of cycles with respect of G and T forms a basis for the cycle space $\mathcal{C}$.

By means of Theorem 1 it is possible to prove Corollary 4

Corollary 4. *Let us assume that the integer min cost flow problem 5.1-5.4 has an unique optimal (basic) feasible solution x^* and let T^* be the spanning tree associated with x^*. Procedure* ***find_feasible_flows*** *examines all the feasible combinations of augmenting cycles starting from x^*.*

Proof. The proof is direct consequence of Theorem 17, that guarantees that all the feasible combinations of augmenting cycles that can be generated by the $m-n+1$ fundamental cycles associated with T^* are examined, and Theorem 1. □

Theorem 18 assures the correctness of the recursive procedure **find_feasible_flows**.

Theorem 18. *Let us assume that the integer min cost flow problem 5.1-5.4 has an unique optimal (basic) feasible solution x^*, then the procedure* ***find_feasible_flows****, starting from x^*, can generate all feasible flows.*

In order to prove Theorem 18, it is necessary to recall a fundamental result related to network flows that has been presented in Chapter 1, that is the **Augmenting Cycle Theorem**.

Theorem 4
Let x and x^0 be two feasible flows. Then x equals x^0 plus the flow on at most m directed cycles in $G(x^0)$. Furthermore the cost of x equals the cost of x^0 plus the cost of the flows on these augmenting cycles.

By means of Corollary 4 and Theorem 4 we can now prove Theorem 18.

(Proof Theorem 18). For the hypothesis, the procedure **find_feasible_flows** starts from the unique optimal (basic) feasible flow x^*. Let G be the graph of the integer min cost flow problem and let T^* be the spanning tree associated with x^*.
According to Theorem 4, a feasible flow x can be obtained from x^* plus at most m augmenting cycles in G.
We shall prove that all flows are generated adding to the initial optimal (basic) feasible flow x^* all the possible feasible combinations of augmenting cycles. Indeed the recursive procedure **find_feasible_flows**, as stated in Corollary 4, is capable of generating all feasible combinations of augmenting cycles starting from an optimal basic feasible flow x^*. Moreover note that the procedure changes only by one unit at a time the flow value. Hence all possible integer feasible flows are generated. □

5.4.2 A special case: degeneracy

It is known that several feasible solutions (flows) could correspond to the same optimal value of the objective function. In these cases, when computing the reduced costs associated with the variables at the initial optimal solution under consideration, some variables not in the basis, will have zero reduced costs. This means that if one of these variables is inserted in the basis, the value of the objective function does not change and the new objective value is equivalent to the optimal one. When this situation occurs, it is necessary to apply the recursive function starting from each of these multiple optimal solutions to obtain all the other worse feasible solutions. For finding all the multiple optimal solutions, starting from one of them that is basic, the same recursive procedure, can be applied, but considering the arcs, out of the basis, with zero reduced costs instead of those with non-zero reduced costs in absolute value. Therefore, in the more general case, including degeneracy, the pseudocode of this procedure is reported below:

INPUT: graph $G = (N, A)$, integer capacities vector u, optimal solution x^*, vector of the non-zero reduced costs in absolute value $\bar{rc}^* = \{rc_1^*, rc_2^*...rc_K^*\}$ (sorted from the smallest to the biggest), vector of the zero reduced costs $\bar{rc}_z^* = \{rc_{z1}^*, rc_{z2}^*...rc_{z_K}^*\}$ and the lists $FS = OPT = \{x^*\}$ containing at the beginning the initial optimal solution.

find_all_feasible_flows($G = (N, A)$, u, x^*, $\bar{rc}^*$, FS)
1: **for** $i = 1$ to $| OPT |$ **do**
2: **find_feasible_flows**($G = (N, A)$, u, x_i^*, $\bar{rc}^*$, FS).
3: **end for**
OUTPUT:Set FS containing all feasible flows.

Algorithm 4: recursive algorithm for the integer min cost flow problem with degeneracy.

The procedure **find_optimal_flows** is the one for generating the multiple optimal solutions that can be pseudocoding in the following way

INPUT: graph $G = (N, A)$, integer capacities vector u, optimal basic feasible solution x^*, vector of the zero reduced costs $\bar{rc}_z^* = \{rc_{z1}^*, rc_{z2}^*...rc_{z_K}^*\}$ and the list $OPT = \{x^*\}$ containing at the beginning the initial optimal basic feasible solution.

find_optimal_flows($G = (N, A)$, u, x^*, $\bar{rc}^*_z$, OPT)

1: $it = 1$
2: **while** $it \leq z_K$ **do**
3: Let $x^*_{(i,j)}$ be the flow variable in the original optimal basic feasible solution x^* corresponding to the element $rc^*_{z_{it}} \in \bar{rc}^*_z$
4: Let C^* be the cycle that the arc (i, j) creates with the arcs belonging to the spanning tree corresponding to the original optimal basic feasible solution x^* in the original network (with capacities vector u)
5: **if** $(x^*_{(i,j)} == u_{(i,j)})$ && $(x^*_{(i',j')} < u_{(i',j')} \forall (i',j') \in C^* : (i,j)$ and (i',j') have opposite orientation) && $(x^*_{(i',j')} > 0 \ \forall (i',j') \in C^* : (i,j)$ and (i',j') have same orientation)) **then**
6: Decrease by one unit the flow and the upper bound on the arc (i, j) and the flow on the arcs $(i',j') \in C^*$ such that (i, j) and (i',j') have same orientation, increase by one unit the flow on the arcs $(i',j') \in C^*$ such that (i, j) and (i',j') have opposite orientation
7: Insert the new optimal feasible flow x^{it} so obtained in OPT
8: **find_optimal_flows**($G = (N, A)$, u^{it}, x^{it}, $\bar{rc}^*_{z_{it}}$, OPT)
9: **end if**
10: **if** $((x^*_{(i,j)} == l_{(i,j)})$ && $(x^*_{(i',j')} < u_{(i',j')} \forall (i',j') \in C^* : (i,j)$ and (i',j') have same orientation) && $(x^*_{(i',j')} > 0 \ \forall (i',j') \in C^* : (i,j)$ and (i',j') have opposite orientation)) **then**
11: Increase by one unit the flow and the lower bound on the arc (i, j) and the flow on the arcs $(i',j') \in C^*$ such that (i, j) and (i',j') have same orientation, decrease by one unit the flow on the arcs $(i',j') \in C^*$ such that (i, j) and (i',j') have opposite orientation
12: Insert the new optimal feasible flow x^{it} so obtained in OPT
13: **find_optimal_flows**($G = (N, A)$, u^{it}, x^{it}, $\bar{rc}^*_{z_{it}}$, OPT)
14: **end if**
15: $it = it + 1$
16: **end while**

OUTPUT: Set OPT containing all equivalent optimal flows.

Algorithm 5: recursive algorithm for finding all optimal flows.

Theorem 19. *Let us assume that the integer min cost flow problem (5.1)-(5.4) has multiple optimal solutions. Let x^* be an optimal basic feasible solution and let T^* be the corresponding spanning tree, then the procedure* ***find_all_feasible_flows****, starting from x^*, can generate all feasible flows.*

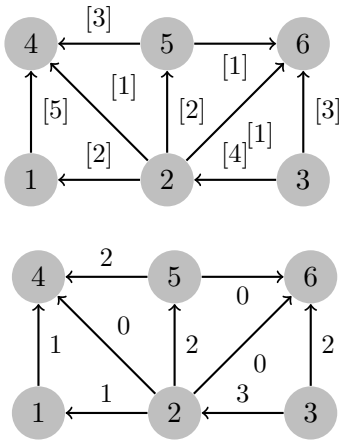
Proof. The proof follows the scheme of Theorem 18 with just one difference: instead of considering only non-zero reduced costs associated with arcs out of T^*, in this case, as degeneracy implies the existence of zero reduced costs also for some arcs out of T^*, the procedure has to consider also these arcs in order to examines all the feasible combinations of augmenting cycles, generated by the $m - n + 1$ fundamental cycles associated with T^*. □

5.4.3 Illustrative example

The following example shows the operating principle of the recursive function for generating all the feasible flows for a directed graph of 6 nodes and 9 arcs. In this case two different flow solutions correspond to the same optimal value of the objective function equal to 34. Then, starting from both these two optimal solutions, the recursive procedure is applied.

Initial optimal solution x^*_1 with objective value equal to 34

We consider the arcs of the network in the following order:



$$\{(1,4);(2,1);(2,4);(2,5);(2,6);(3,2);(3,6);(5,4);(5,6)\}$$

Optimal solution x_1^*, Dual variables (Potentials) and Reduced costs

$$x^* = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 2 \\ 0 \\ 3 \\ 2 \\ 2 \\ 0 \end{bmatrix} \quad \pi^* = \begin{bmatrix} 1 \\ -1 \\ -5 \\ 3 \\ 1 \\ 1 \end{bmatrix} \quad \bar{rc}^* = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ -4 \\ 0 \\ 0 \\ 0 \\ -2 \end{bmatrix}$$

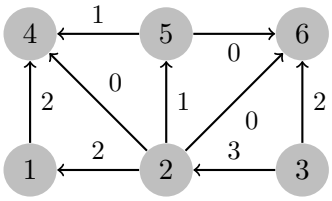
(In this example the reduced costs have been computed assuming positive reduced costs for saturated arcs and negative reduced costs for empty arcs out of the basis).

Procedure applied on the example

- The variable with minimum non-zero reduced cost (in absolute value) is $x^*_{(2,5)}$.
- Decrease by one unit the flow and the upper bound on the arc $(2,5)$.
- If this arc entered in the basis, the cycle C would consist in the arcs $(2,1)$, $(1,4)$, $(2,5)$ and $(5,4)$.
- Increase by one unit the flow on the arcs $(2,1)$ and $(1,4)$, and decrease by one unit the flow on the arc $(5,4)$.

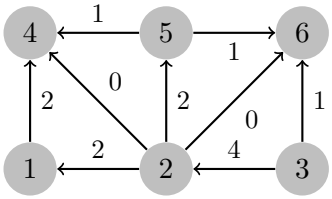
  Second best flow x^1 with objective value equal to 35

  - The arc $(2,1)$ is now saturated.
  - Consider the original network and the original optimal solution.



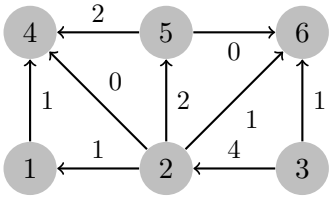
– The variable with the second minimum non-zero reduced cost (in absolute value) is $x^*_{(5,6)}$
– If this variable entered in basis, the cycle C would consist in the arcs $(2,1)$, $(1,4)$, $(3,2)$, $(3,6)$, $(5,4)$ and $(5,6)$.
– Increase by one unit the flow and the lower bound on the arc $(5,6)$.
– Increase by one unit the flow on the arcs $(1,4)$ and $(2,1)$ and $(3,2)$, and decrease by one unit the flow on the arcs $(3,6)$ and $(5,4)$.

New feasible solution x^2 with objective value equal to 36



– The arc (2,1) is now saturated.
– Consider the original network and the original optimal solution.
– The variable with the third minimum non-zero reduced cost (in absolute value) is $x^*_{(2,6)}$
– If this variable entered in basis, the cycle C would consist in the arcs $(2,6)$, $(3,2)$ and $(3,6)$.
– Increase by one unit the flow and the lower bound on the arc $(2,6)$.
– Increase by one unit the flow on the arc $(3,2)$, and decrease by one unit the flow on the arc $(3,6)$.

New feasible solution x^3 with objective value equal to 38



– Consider again the variable with minimum non-zero reduced cost (in absolute value), that is $x^*_{(2,5)}$.

– Decrease by one unit the flow and the upper bound on the arc $(2, 5)$.
– As before, if this arc entered in the basis, the cycle C would consist in the arcs $(2, 1)$, $(1, 4)$, $(2, 5)$ and $(5, 4)$.
– Increase by one unit the flow on the arcs $(2, 1)$ and $(1, 4)$, and decrease by one unit the flow on the arc $(5, 4)$.

New feasible solution x_1^3 with objective value equal to 39

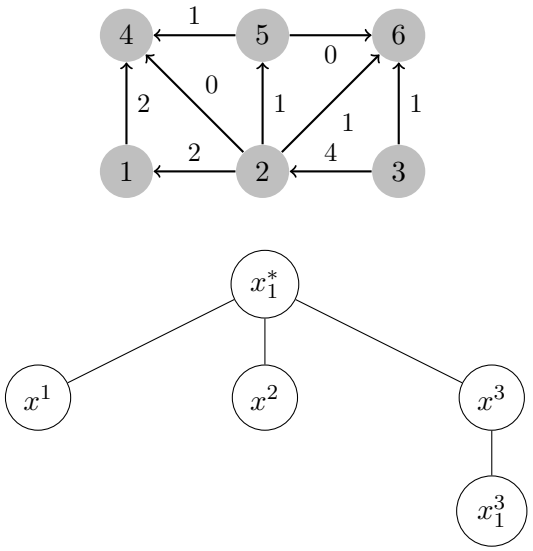
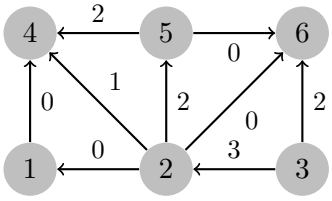


Figure 5.3. Solutions tree from the optimal basic feasible solution x_1^*

At this point, all the feasible flows starting from the first optimal basic feasible solution were generated. Therefore, the recursive procedure can be applied starting from the second optimal solution.

Initial optimal solution x_2^* with objective value equal to 34

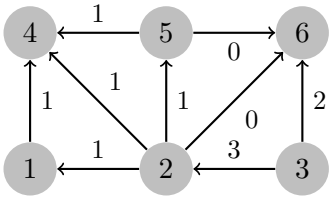


Optimal solution x_2^*, Dual variables (Potentials) and Reduced costs

$$x^* = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 2 \\ 0 \\ 3 \\ 2 \\ 2 \\ 0 \end{bmatrix} \quad \pi^* = \begin{bmatrix} 1 \\ -1 \\ -5 \\ 3 \\ 1 \\ 1 \end{bmatrix} \quad \bar{rc}^* = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ -4 \\ 0 \\ 0 \\ 0 \\ -2 \end{bmatrix}$$

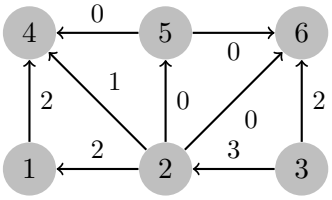
- The variable with minimum non-zero reduced cost (in absolute value) is $x^*_{(2,5)}$.
- Decrease by one unit the flow and the upper bound on the arc $(2, 5)$.
- If this arc entered in the basis, the cycle C would consist in the arcs $(2, 1)$, $(1, 4)$, $(2, 5)$ and $(5, 4)$.
- Increase by one unit the flow on the arcs $(2, 1)$ and $(1, 4)$, and decrease by one unit the flow on the arc $(5, 4)$.

Second best flow x^1 with objective value equal to 35



Procedure applied on the example

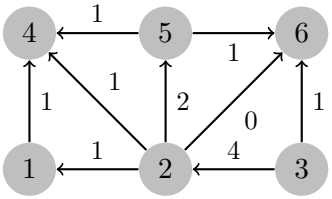
- Consider again the variable with minimum non-zero reduced cost (in absolute value), that is $x^*_{(2,5)}$.
- Decrease by one unit the flow and the upper bound on the arc $(2, 5)$.
- As before, if this arc entered in the basis, the cycle C would consist in the arcs $(2, 1)$, $(1, 4)$, $(2, 5)$ and $(5, 4)$.
- Increase by one unit the flow on the arcs $(2, 1)$ and $(1, 4)$, and decrease by one unit the flow on the arc $(5, 4)$.



New feasible solution x_1^1 with objective value equal to 36

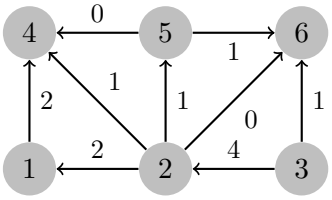
- The arc (2,1) is now saturated.
- Consider the original network and the original optimal solution.
- The variable with the second minimum non-zero reduced cost (in absolute value) is $x^*_{(5,6)}$
- If this variable entered in basis, the cycle C would consist in the arcs $(2,1)$, $(1,4)$, $(3,2)$, $(3,6)$, $(5,4)$ and $(5,6)$.
- Increase by one unit the flow and the lower bound on the arc $(5,6)$.
- Increase by one unit the flow on the arcs $(1,4)$ and $(2,1)$ and $(3,2)$, and decrease by one unit the flow on the arcs $(3,6)$ and $(5,4)$.

New feasible solution x^2 with objective value equal to 36



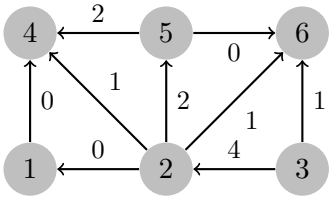
- Consider again the variable with minimum non-zero reduced cost (in absolute value), that is $x^*_{(2,5)}$.
- Decrease by one unit the flow and the upper bound on the arc $(2,5)$.
- As before, if this arc entered in the basis, the cycle C would consist in the arcs $(2,1)$, $(1,4)$, $(2,5)$ and $(5,4)$.
- Increase by one unit the flow on the arcs $(2,1)$ and $(1,4)$, and decrease by one unit the flow on the arc $(5,4)$.

New feasible solution x_1^2 with objective value equal to 37



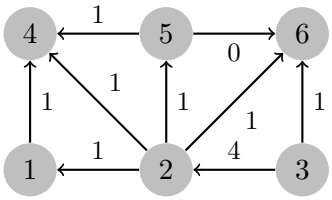
– The arc (2,1) is now saturated.
– Consider the original network and the original optimal solution.
– The variable with the third minimum non-zero reduced cost (in absolute value) is $x^*_{(2,6)}$
– If this variable entered in basis, the cycle C would consist in the arcs $(2,6)$, $(3,2)$ and $(3,6)$.
– Increase by one unit the flow and the lower bound on the arc $(2,6)$.
– Increase by one unit the flow on the arc $(3,2)$, and decrease by one unit the flow on the arc $(3,6)$.

New feasible solution x^3 with objective value equal to 38



– Consider again the variable with minimum non-zero reduced cost (in absolute value), that is $x^*_{(2,5)}$.
– Decrease by one unit the flow and the upper bound on the arc $(2,5)$.
– As before, if this arc entered in the basis, the cycle C would consist in the arcs $(2,1)$, $(1,4)$, $(2,5)$ and $(5,4)$.
– Increase by one unit the flow on the arcs $(2,1)$ and $(1,4)$, and decrease by one unit the flow on the arc $(5,4)$.

New feasible solution x^3_1 with objective value equal to 39



– Consider again the variable with minimum non-zero reduced cost (in absolute value), that is $x^*_{(2,5)}$.
– Decrease by one unit the flow and the upper bound on the arc $(2,5)$.
– As before, if this arc entered in the basis, the cycle C would consist in the arcs $(2,1)$, $(1,4)$, $(2,5)$ and $(5,4)$.
– Increase by one unit the flow on the arcs $(2,1)$ and $(1,4)$, and decrease by one unit the flow on the arc $(5,4)$.

New feasible solution $x^3_{1,1}$ with objective value equal to 40

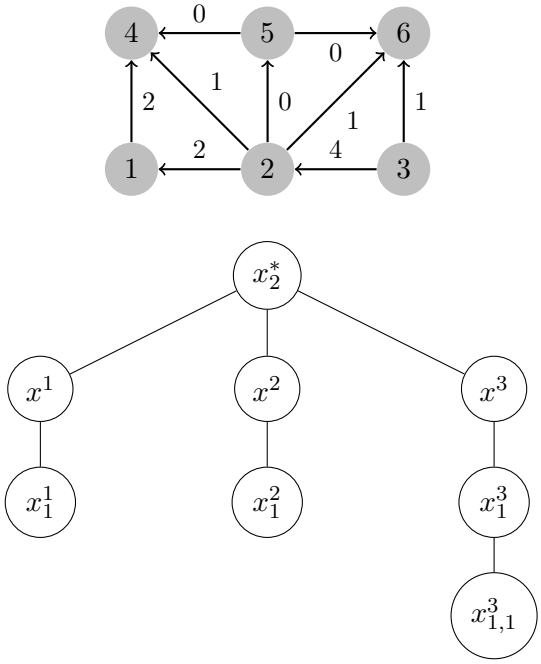


Figure 5.4. Solutions tree from the optimal solution x_2^*

5.4.4 Adaptation of the recursive procedure to the second phase

The recursive algorithm described in Section 5.4.1, 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 presented in the previous section is applied setting an upper bound on the value of the objective function for the weighted sum problem. For the first triangle this upper bound is defined (see [70]) as follows:

$$\Delta = \lambda_1(y_1(x_{i+1}) - 1) + \lambda_2(y_2(x_i) - 1) \tag{5.6}$$

This upper bound is updated during the recursive procedure by means of the new flows generated. In more details, when a new feasible flow is produced, the corresponding point in the objective space is computed. If this point is non-dominated by the other points already stored in the set of non-dominated points, and belongs to the current triangle, the corresponding flow is inserted in the list of the "potentially efficient solutions" and is used to update the upper bound by means of the formula below (see [69]):

$$\Delta = \max\{\lambda_1(y_1(x_{i,j+1}) - 1) + \lambda_2(y_2(x_{i,j}) - 1),\ \ j = 0, ..., r\} \tag{5.7}$$

where r is the number of efficient solutions found in the triangle under consideration and it is assumed that $x_{i,0}, x_{i,1}, ...x_{i,r+1}$ 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 supported solutions associated with the triangle). Moreover, if these conditions are satisfied, the network associated with the new flow is used for generating other points. In more details, the single objective problem on the new network in which the new flow is optimal, is again solved by means of the dual variant of Benson's algorithm. New points are produced. Some of these may belong to the triangle under consideration and used to update the upper bound for this triangle. The remaining points may lie on triangles not yet explored and for this reason are stored and used for updating the upper bound in the next iterations.
The algorithm terminates when all the triangles are explored by means of this recursive procedure. The output of the algorithm is a complete set of efficient solutions. The pseudocode of the whole algorithm is shown below:

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

Two_phase_bflow

$i = 1$
$\mathcal{E} = \emptyset$
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
    $\Delta = \max\{\lambda_1(y_1(x^{i,j+1}) - 1) + \lambda_2(y_2(x^{i,j}) - 1), j = 0, ..., r\}$, $r + 1 =\mid \mathcal{E}_i \mid$
  end if
else
  $\Delta = \mu_\lambda = \lambda_1(y_1(x^{i+1}) - 1) + \lambda_2(y_2(x^i) - 1)$ /* initial value of Delta*/
  $\mathcal{E}$ = **find_all_feasible_solutions**$[N, u, x^i, \bar{rc}^i, \mathcal{E}, \Delta]$. /*with $\bar{rc}^i$ vector of the non-zero reduced costs in absolute value associated to x^i (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 $\mathcal{E}$
      $\mathcal{E}_b$ = **Bensolve**$[N^b]$
    end if
  end for
  for $x^b \in \cup_b \mathcal{E}_b$ **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 $\mathcal{E}_i$
    end if
  else
    Insert x^b in $\mathcal{E}$
  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.

Algorithm 6: Two-phase algorithm for the bi-objective min cost flow problem.

Theorem 20. *The set $\mathcal{E}^* = \cup_{i=1,...,s-1}\mathcal{E}_i$ generated by the algorithm **Two_phase_bflow** is a complete set of efficient solutions of the bi-objective mini-*

mum cost flow problem.

Proof. Without loss of generality we assume $1 \leq i \leq s-1$.
$\mathcal{E}_i$ contains all 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 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 would be 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.
$\mathcal{E}_i$ contains a complete set of efficient solutions within the triangle T_i.
$\mathcal{E}_i$ 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 $\mathcal{E}_i$. The final step of the algorithm checks if in $\mathcal{E}_i$ some non-efficient solutions exist and remove them. Then when the algorithm ends $\mathcal{E}_i$ contains a complete set of efficient solutions. □

5.5 Preliminary Results

In this section we analyze the complete sets of non-dominated points produced by the algorithm on some instances taken from the literature. For the first phase of the algorithm we used the software Bensolve (see [80]), a solver for

vector linear programs, which includes the subclass of multiple objective linear programs. Bensolve is an open source implementation of Benson's algorithm and its dual variant. Both algorithms compute primal and dual solutions of vector linear programs. Bensolve requires the VLP format as input file, that is an extension of the GLPK LP format to the case of multiple objective linear programs and vector linear programs. The computational results are provided through different files. For our purpose we used those in which are written line-wise the vertices and extreme directions of the upper image, the primal solutions and the basis information of the primal problem respectively. As regards the second phase the algorithm has been implemented in C++ by means of the Xcode (Integrated development environment for macOS Apple developers). As our main objective was to verify the correctness of the approach, this first implementation does not use Bensolve in the second phase. The instances on which the algorithm has been tested, have been selected out of the set of test instances presented in [70]. We chose randomly two instances from each of the sets F01-F02 and G01-G02 used in [70]. The sets F01-F02 consist of problem instances generated by NETGEN (see [59]), with some modifications to include a second objective, with fixed total sum of the supply equal to 100, maximum arc capacity equal to 50 and maximum arc cost equal to 100, as summarized in Table 5.1. The sets G01-G02 contains instances with grid structure, that is graphs that can be drawn as a grid with a given height h and width w. These instances are characterized, as the sets F01-F02, by maximum arc capacity equal to 50 and maximum arc cost equal to 100 and they also have fixed total sum of the supply equal to 100 (see Table 5.2). We tested the algorithm also on the sets N01-N02 proposed in [70], that is the first two sets of instances generated by NETGEN varying the total sum of the supply. However, as the first two sets have the same characteristics of the sets F01-F02 (same dimensions and total sum of the supply is equal to 100) here we report only the tests related to the instances randomly selected from F01-F02.

Table 5.1. NETGEN Instances

Set Name	n	m	Sources	Sinks
F01	20	60	9	7
F02	20	80	9	7

Table 5.2. Grid Instances

Set Name	h	w	n	m
G01	4	5	20	62
G02	5	8	40	134

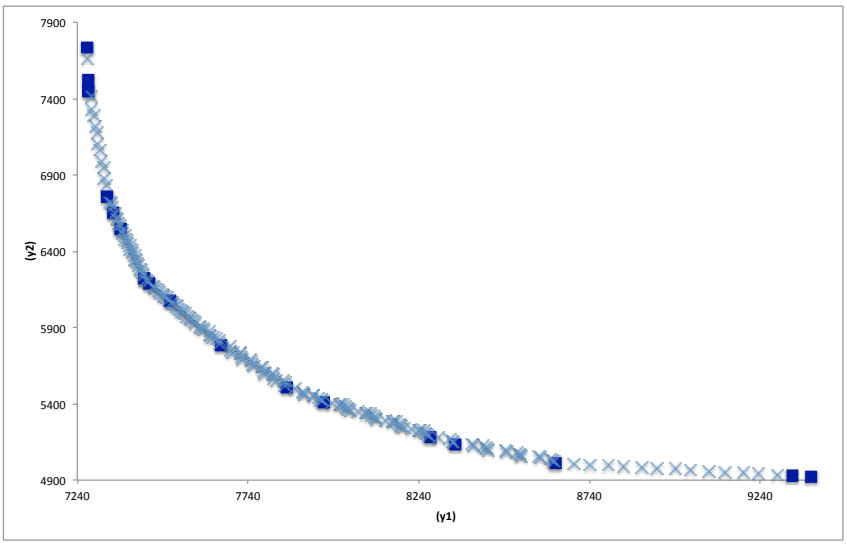


Figure 5.5. Pareto Frontier for Instance 1-F01

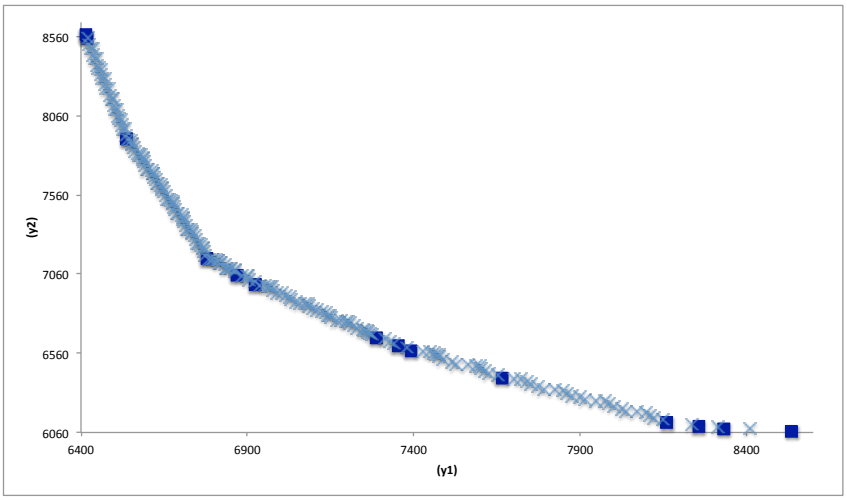


Figure 5.6. Pareto Frontier for Instance 2-F01

Table 5.3. Summary on the instances from F01-F02

Instance number	$\mid Y_N \mid$	$\mid Y_{SN} \mid$	$\mid Y_{NN} \mid$	$\frac{\mid Y_{SN} \mid}{\mid Y_{NN} \mid}$
1-F01	190	53	137	0.4
2-F01	200	36	164	0.2
1-F02	204	38	166	0.2
2-F02	197	57	140	0.4

Figures 5.5, 5.6, 5.7 and 5.8 show the Pareto frontier of the instances from F01 and F02. The non-dominated points corresponding to extreme efficient solutions, found in the first phase by means of Bensolve, are colored in blue.

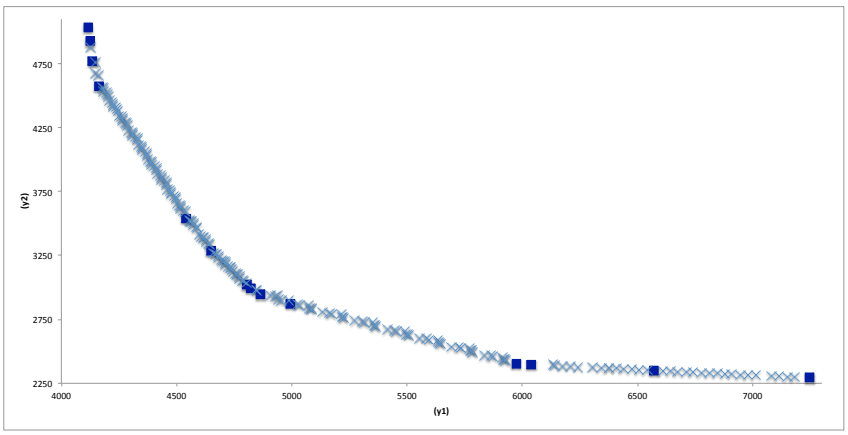


Figure 5.7. Pareto Frontier for Instance 1-F02

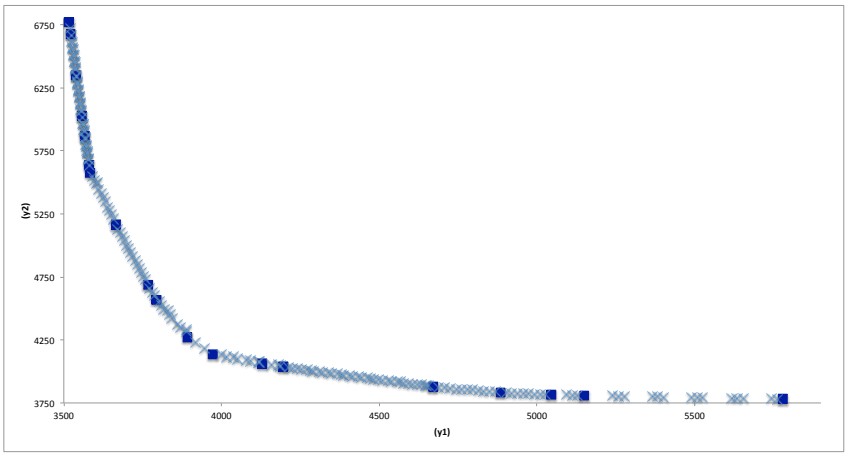


Figure 5.8. Pareto Frontier for Instance 2-F02

The remaining non-dominated points, represented in sky blue, have been generated through the second phase of the algorithm. In this latter set of points, there are non-supported points but also supported points found during the recursive procedure applied considering arcs out of the basis with zero reduced costs. Indeed, in all these instances we are in presence of degeneracy. In this situation the recursion first generates the multiple solutions of the single objective problem associated with a given triangle and then the flows corresponding to worse values of the objective function, as described in Section 5.4.1. As already noticed on averages values in [70], we can make the following observations from Table 5.3:

- The ratio of supported and non-supported non-dominated points, which ranges between 0.2 and 0.4 for the instances randomly selected, indicates that the majority of the non-dominated points are non-supported.
- The ratio of supported and non-supported non-dominated points decreases when the total number of non-dominated points increases.

– Increasing the number of arcs and fixing the number of nodes, the number of non-dominated points increases.

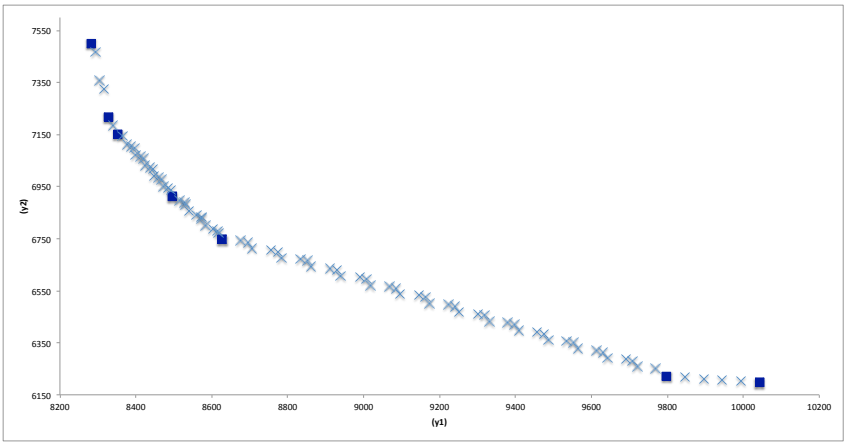


Figure 5.9. Pareto Frontier for Instance 1-G01

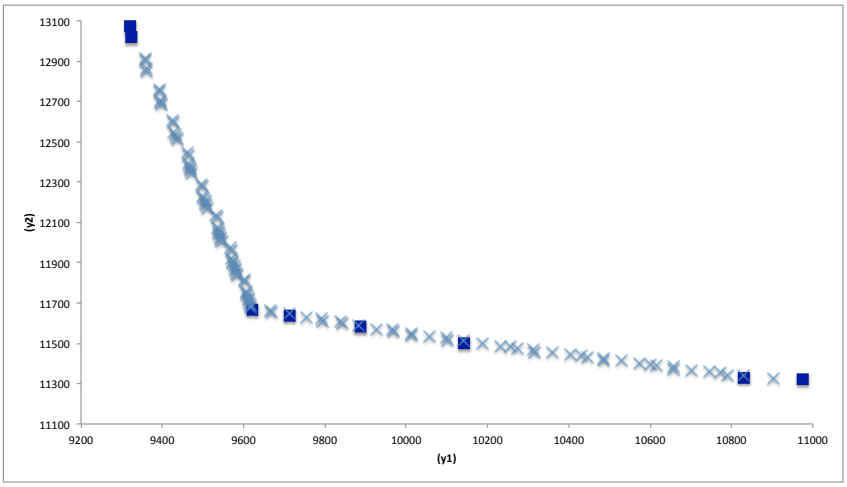


Figure 5.10. Pareto Frontier for Instance 2-G01

Table 5.4. Summary on the instances from G01-G02

Instance number	$\mid Y_N \mid$	$\mid Y_{SN} \mid$	$\mid Y_{NN} \mid$	$\frac{\mid Y_{SN} \mid}{\mid Y_{NN} \mid}$
1-G01	85	34	51	0.7
2-G01	109	21	88	0.2
1-G02	237	35	202	0.2
2-G02	131	35	96	0.4

Figures 5.9, 5.10, 5.11 and 5.12 show the Pareto frontier of the instances from G01 and G02. As for the instances from F01 and F02, the non-dominated points corresponding to extreme efficient solutions, are colored in blue, while

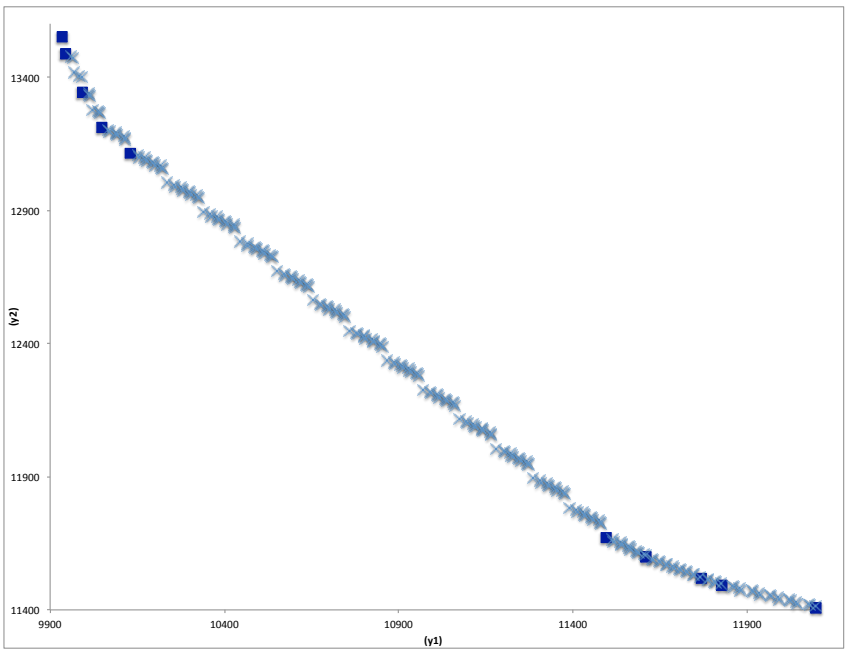


Figure 5.11. Pareto Frontier for Instance 1-G02

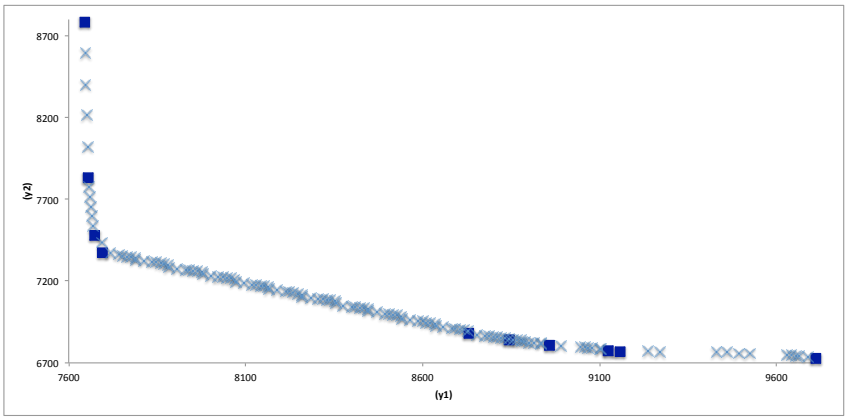


Figure 5.12. Pareto Frontier for Instance 2-G02

the remaining non-dominated points are represented in sky blue. Looking at the ratio of supported and non-supported non-dominated points, ranging between 0.2 and 0.7, we can observe that also for the grid instances selected, the majority of the non-dominated points are non-supported and that this ratio decreases when the total number of non-dominated points decreases. Eventually, for most of the instances here reported, and also for most of the instances on which the algorithm has been tested at this stage of the implementation, we can notice that the non-supported non-dominated points are located very close to the boundary of $conv(Y)$. However, there exist some exceptions in which non-dominated non-supported points are far from the boundary of $conv(Y)$ as we can observe in Figure 5.13 representing all the non-dominated points in the second triangle of the grid instance shown in Figure 5.10.

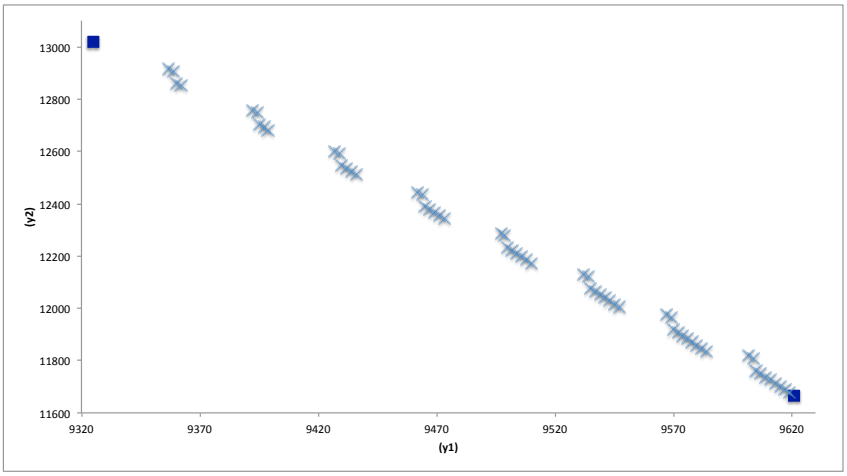


Figure 5.13. Portion of the Pareto Frontier for Instance 2-G01

In conclusion, by means of these tests, it has been verified the exactness of the approach proposed, so encouraging the possibility to make improvements from a computational point of view. Then, the next stage will consist in working on a more efficient implementation also with the use of Bensolve in the second phase, as described in Section 5.4.

Chapter 6

A new algorithm for the bi-objective minimum spanning tree problem

In this chapter we present a new two-phase algorithm for determining a complete set of efficient solutions for the bi-objective minimum spanning tree problem, described in [7]. In the first phase the extreme efficient solutions are generated. In the second one the solutions found in the previous phase are used to limit the exploration area in the outcome set for finding the remaining non-supported efficient solutions.

6.1 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 the multi-objective combinatorial optimization. In [26] a generalization of Prim's algorithm is presented. The idea of this generalization is to build trees by covering one additional vertex at each iteration by selecting efficient edges in the subset connecting covered with non-covered vertices. However in this procedure also non-efficient spanning trees may be generated. In [48] an enhancement of Corley's algorithm, consisting in deleting at each iteration subtrees which are non-efficient, is proposed. The resulting algorithm is also presented in [31] and used in [85] to evaluate a genetic algorithm able to approximate the Pareto frontier proposed by the authors. However in [60] is showed that some efficient spanning trees may be not generated and consequently that also some non-efficient spanning trees may be produced by this procedure. In [74] a generalization of Kruskal's algorithm was proposed. Also in [67] 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.

As regards the generic approaches for multi-objective combinatorial optimization both two-phase methods and multi-objective Branch&Bound have been studied. In [71] a two-phase approach for the bi-objective minimum spanning tree problem is presented, that uses a Branch&Bound algorithm in the second phase. 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. From the computational results of this work comes to light that this approach may not be practical for big instances. In [77] the authors propose another two-phase approach based on a ranking algorithm, the k-best algorithm by Gabow [39], applied in the second phase to explore the triangles identified by the extreme non-dominated points given by the first phase. They provided also a comparison with the procedure by Ramos that shows the superiority of their approach in term of computational time and instances size that are solvable. Moreover, an heuristic enhancement of the k-best algorithm was proposed that generally speeds-up the running time. In the most recent work by [76], an improved Branch&Bound algorithm was 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, exceeding also the algorithm proposed in [77] in terms of instances size that are solvable. More recently, in [35], the authors present a comparison among alternative reinforced formulations and new formulations for the minimum spanning tree problem, also with more than two objectives, considering a specific form of aggregation of the criteria, called Ordered Weighted Average operator, in order to generate supported efficient solutions of the problem. They showed that an appropriate formulation allows to solve larger instances and with more objectives than those previously solved in the literature.

6.2 Algorithm description

For the first phase of the algorithm proposed in this work for the bi-objective minimum spanning tree problem, we selected one of the standard techniques for multi-objective linear programming problems, that is the weighted sum method (see [32]). In this context, this method can be used for generating only the supported efficient solutions because of the integrality constraints on the decisional variables (see [33]) and for this specific problem it is applied by means of the single commodity flow formulation of the MST problem, described in Section 6.3. Another approach that has been tried for generating supported efficient solutions, is the dual variant of Benson's algorithm. As mentioned in Chapter 5, this is a multi-objective linear programming solution method that can be applied also in the integer case if the integrality property

is satisfied. For this reason, the Kipp-Martin formulation of the minimum spanning tree problem (see [58]) was considered. This is a linear formulation of the minimum spanning tree problem that assures the integrality of its (binary) solutions. In the second phase, as for the bi-objective integer min cost flow problem, we considered consecutive pairs of non-dominated points associated to extreme efficient solutions, sorted in increasing order with respect to the first objective. From each of these pairs, the exploration area is defined by the triangle whose vertices are the two non-dominated points of the pair and the corresponding Local Nadir Point. Then, in each triangle, a recursive procedure, that will be illustrated in Section 6.4.1, is applied, starting from one of the two extreme efficient solutions associated with that triangle, in order to generate non-supported efficient solutions. At each call of the recursive algorithm, if the spanning tree produced corresponds to a non-dominated point that belongs to the triangle under consideration, the upper bound in the current triangle is updated (see [69]). The reiteration of this procedure in each triangle generates a complete set of efficient solutions.

6.3 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 MST problem. The first one is the weighted sum method which consists in assigning a weight for each objective and solving the single objective problem given by the weighted sum of the objectives. As explained in Chapter 3, solving iteratively the single objective problem, by varying each weight between 0 and 1, it is possible to generate all the supported efficient solutions of an integer linear multi-objective problem. The details of this approach are explained in Section 6.3.1 in relation with the flow formulation of the MST problem. The second method adopted for the first phase is the dual variant of Benson's algorithm. To apply it on the MST problem, it was necessary to consider a different formulation of the problem, that is the Kipp-Martin formulation which is a linear formulation characterized by the integrality of the binary solutions. Details on this formulation and its properties will be given in Section 6.3.2.

6.3.1 The weighted sum method on the flow formulation

As mentioned above, the weighted sum method used in the first phase of the two-phase algorithm, was applied on the flow formulation of the MST problem (see [64]) reported below.

Flow formulation of the MST problem

$$\min \sum_{e \in E} c_e x_e \tag{6.1}$$

subject to

$$\sum_{j:(1,j)\in E} f_{1j} - \sum_{j:(j,1)\in E} f_{j1} = n-1 \tag{6.2}$$

$$\sum_{j:(j,i)\in E} f_{ji} - \sum_{j:(i,j)\in E} f_{ij} = 1,\ \forall i \in V,\ i \neq 1 \tag{6.3}$$

$$f_{ij} \leq (n-1)x_e\ \forall e = (i,j) \in E \tag{6.4}$$

$$f_{ji} \leq (n-1)x_e\ \forall e = (i,j) \in E \tag{6.5}$$

$$\sum_{e\in E} x_e = n-1 \tag{6.6}$$

$$f_{ij}, f_{ji} \geq 0\ \forall e = (i,j) \in E \tag{6.7}$$

$$x_e \in \{0,1\}\ \forall e \in E \tag{6.8}$$

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 (6.2) and the rest of the network nodes are sinks requiring each one one unit of flow (6.3). 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 solution. The flow f_{ij} and f_{ji} can be non-zero only if the corresponding edge is in solution and is limited by the total flow from the source, equal to $n-1$ (6.4)-(6.5). Eventually, the total number of edges activated has to be equal to $n-1$ (6.6) 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. To this end, the single objective problems obtained considering one objective at a time, are solved ($\lambda = 1$ and $\lambda = 0$ respectively). Then the solutions so obtained are compared. If they are different, two new intervals for λ are identified splitting in two parts of equal dimension the original one ($[0,1]$). On each of these new intervals the same procedure is repeated: the single objective problems obtained taking λ equal to each of the extreme values of the intervals, are solved. If the solutions obtained at the extreme of the single interval are different a new split is made and so on. This procedure runs until the dimension of the intervals generated is greater or equal to a given bound *md* defined a priori. The pseudocode of the method applied on the flow formulation for the MST problem is listed below.

Recursive function for implementing the weighted sum method for the bi-objective MST problem

INPUT FIRST CALL: Network $G = (V, E)$, initial value of λ, initial interval [ex_1,ex_2] ([0,1] at the first call of the function) and the list of supported efficient solutions Sol (containing at the beginning the solutions associated with λ equal to 0 and 1).

weighted_sum_method($G = (V, E)$, λ, ex_1, ex_2, Sol)
1: **if** $(ex_2 - ex_1) \geq md$ **then**
2: solve_model($G = (V, E)$, λ, Sol)
3: **if** (($Sol(\lambda) = Sol(ex_1)$ && ($Sol(\lambda) \neq Sol(ex_2)$) /*$Sol(\lambda)$ indicates the solution obtained using the weights (λ, $1 - \lambda$) respectively for the first and the second objective*/ **then**
4: weighted_sum_method($G = (V, E)$,$1 - \frac{(ex_2 - \lambda)}{2}$, λ, ex_2, Sol)
5: **end if**
6: **if** (($Sol(\lambda) \neq Sol(ex_1)$ && ($Sol(\lambda) = Sol(ex_2)$) **then**
7: weighted_sum_method($G = (V, E)$,$\frac{(\lambda - ex_1)}{2}$, ex_1, λ, Sol)
8: **end if**
9: **if** (($Sol(\lambda) \neq Sol(ex_1)$ && ($Sol(\lambda) \neq Sol(ex_2)$) **then**
10: weighted_sum_method($G = (V, E)$,$1 - \frac{(ex_2 - \lambda)}{2}$, λ, ex_2, Sol)
11: weighted_sum_method($G = (V, E)$,$1 - \frac{(\lambda - ex_1)}{2}$, ex_1, λ, Sol)
12: **end if**
13: **end if**
14: return

OUTPUT: Set Sol containing all supported efficient solutions.

Algorithm 7: weighted sum algorithm.

6.3.2 Benson's algorithm on the Kipp-Martin formulation

An alternative approach for implementing the first phase, as mentioned in Section 6.3, is represented by the use of the dual variant of Benson's algorithm. As described in Chapter 3, 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. In the integer case, this algorithm can be applied only if the integrality property of the solutions is guaranteed. In [58] a linear programming formulation for the MST problem, obtained by adding auxiliary variables, 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, it is sure that the solutions so obtained will be integer, more precisely binary. In this formulation, reported below, new non-negative variables z_{kij}, associated with triad 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

$$\min \sum_{e \in E} c_e x_e \tag{6.9}$$

subject to

$$\sum_{e \in E} x_e = n - 1 \tag{6.10}$$

$$z_{kij} + z_{kji} = x_e \;\; k = 1, ..., n, \;\; e \in \gamma(\{i, j\}) \tag{6.11}$$

$$\sum_{s>i} z_{kis} - \sum_{h<i} z_{kih} \leq 1 \;\; k = 1, ..., n, \;\; i \neq k \tag{6.12}$$

$$\sum_{s>k} z_{kks} - \sum_{h<k} z_{kkh} \leq 0, \;\; k = 1, ...n \tag{6.13}$$

$$x_e \geq 0 \;\; \forall e \in E \tag{6.14}$$

$$z_{kij} \geq 0 \;\; \forall k, i, j \tag{6.15}$$

Constraints (6.11), (6.12) and (6.13) guarantee that the binary solution does not contain undirected or directed cycles and constraint (6.10) assures that the binary solution is a spanning tree.

6.4 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 performed 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 their associated Local Nadir point. At each iteration of the algorithm, one of these triangles is analyzed, starting from the left, that is, from the triangle 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 spanning tree associated with the extreme efficient solution on the right of the triangle (represented by the values of the binary variables associated to the arcs of the network), is used as initial optimal solution for the single objective minimum spanning tree problem. To be more precise we report below a standard formulation of the MST problem, however at implementation level any formulation with the objective function defined as in (6.16) is suitable.

$$\min \sum_{e \in E} (\lambda_1 c_e^1 x_e + \lambda_2 c_e^2 x_e) \tag{6.16}$$

subject to

$$\sum_{e \in E} x_e = n - 1 \tag{6.17}$$

$$\sum_{e \in E(S)} x_e \leq \mid S \mid -1 \;\; \forall S \subseteq N, \;\; S \neq \emptyset \tag{6.18}$$

$$x_e \in \{0, 1\} \quad \forall e \in E \tag{6.19}$$

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. These weights are defined as in [70]: 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 defined as follows:

$$\lambda_1 = y_2(x^i) - y_2(x^{i+1}) \quad \text{and} \quad \lambda_2 = y_1(x^{i+1}) - y_1(x^i). \tag{6.20}$$

This definition of weights implies that the single objective MST problem has optimal solutions x^i and x^{i+1}. Hence, it is possible to start from the solution x^{i+1} (or from the solution x^i) and consider it optimal for the weighted sum problem (6.16)-(6.19). Then, from this solution, a recursive algorithm, described in Section 6.4.1, able to generate all the spanning trees of the single objective MST problem is applied. Indeed, all the other efficient solutions of the original bi-objective problem, correspond to spanning trees for the network under consideration such that the associated points in the objective space are non-dominated. For this reason a new algorithm that provides all the spanning trees of a given network, starting from an initial optimal solution, has been designed. This algorithm, differently from the k-best algorithm (see [39]), does not produce spanning trees in strict ranking order with respect to the objective function but, according to its operating principle, at each iteration it generates a spanning tree that is certainly worse than the previous one in term of objective value.

6.4.1 Procedure for generating all the spanning trees of a graph

The procedure used 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 to the biggest one in absolute value and analyzed in this order. Given the smallest non-zero reduced cost in absolute value rc_e^*, the fundamental cycle C_e^* that the corresponding edge e creates with 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 means of the exchange of 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 spanning trees so obtained are progressively worse with respect to the value of the objective function.
At the first iteration of this algorithm, after the first exchange, the spanning tree obtained is exactly the second best spanning tree generated inserting in the initial optimal tree, the edge with minimum non-zero reduced cost in absolute value and removing from the fundamental cycle so created, the edge with maximum cost. Therefore, the corresponding deterioration of the objective function is as minimum as possible. When all the arcs in the fundamental cycle created by the arc e are exchanged with e, the second non-zero reduced cost in absolute value has to be considered, starting again from the original optimal spanning tree, the same from which the second best spanning tree was generated.
Considering the second reduced cost in absolute value $e^{'}$, 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, in order not to miss feasible solutions and to obtain the minimum possible deterioration of the objective function, starting from the new spanning tree $T^{'}$ obtained, the first non-zero reduced cost in absolute value has to be considered implementing again all the possible exchanges of the edge e with the edges in the fundamental cycle created by e in the current spanning tree $T^{'}$. Iterating this procedure considering all the non-zero reduced costs in absolute value in order from the smallest to the biggest, and for all their possible combinations, as explained, all the spanning trees can be identified. Therefore the resulting algorithm consists in a recursive procedure that is summarized in the following pseudocode.

Recursive function for finding all spanning trees of a graph

INPUT: graph $G = (N, E)$, costs vector c, optimal solution T^*, vector of the non-zero reduced costs in absolute value $\bar{rc}^* = \{rc_1^*, rc_2^*...rc_K^*\}$ (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^*, $\bar{rc}^*$, FT)
1: $it = 1$
2: **while** $it \leq K$ **do**
3: Let e be the edge out of T^* corresponding to the element $rc_{it}^* \in \bar{rc}$ (c_1^* is the minimum non-zero reduced cost in absolute value)
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 biggest to the smallest) 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: Add the tree $T^{'}$ so obtained to the list FT
9: **if** $it > 1$ **then**
10: find_all_spanning_trees($G = (N, E)$, c, $T^{'}$, $it - 1$, FT)
11: **end if**
12: **end for**
13: $it = it + 1$
14: **end while**
OUTPUT: Set FT containing all spanning trees of the graph G.

Algorithm 8: recursive algorithm specialized for the minimum spanning tree problem.

Theorem 21. *Let us assume that the minimum spanning tree problem (6.16)-(6.19) has an unique optimal solution T^*. Procedure* ***find_feasible_trees*** *examines all the feasible combinations of T^*-exchanges.*

Proof. To prove this we just recall that 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 (in absolute value). Edges of this fundamental cycle are sorted by cost (from the biggest to the smallest) and 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 starts 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 (in absolute value) and sorts its edges by cost. Then it generates just one T^*-exchange between the edge under consideration and the first edge in the fundamental cycle. Now, differently from what done in the case of the first non-zero reduced cost (in absolute value), 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 (in absolute value) with the first T^*-exchange associated with the first non-zero reduced cost (in absolute value). Note that continuing in generating T^*-exchanges associated with the edge related to the first non-zero reduce cost (in absolute value), all the T^*-exchanges from the first non-zero reduced cost (in absolute value) 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 in absolute value). 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 (in absolute value), the procedure seeks a further T^*-exchange from the same fundamental cycle. If it succeeded, again the recursive procedure is called from the new feasible tree so generated. It should be clear that by means of this mechanism, all the feasible combinations of T^*-exchanges generated by the first two fundamental cycles are examined. At the generic stage of the algorithm non-zero reduced cost 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 (in absolute value) with the T^*-exchanges associated with non-zero reduced cost k 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 order to prove the correctness of the recursive procedure **find_feasible_trees**, we recall Theorem 2 from Chapter 1.

Theorem 2
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 the tree T by at most $D(T,T^{'})$ cyclic interchanges, where:

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

Theorem 22. *Let T^* be the unique optimal solution of the minimum spanning tree problem (6.16)-(6.19) 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 (6.16)-(6.19).

According to Theorem 2, 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 21, generates all feasible combinations of T^*-exchanges. Hence all the other spanning trees are produced. □

Observation 2. *It is worth to observe that in this way it is possible to generate more than once the same spanning tree.*

6.4.2 The recursive procedure adapted to the second phase

The recursive algorithm described in Section 6.4.1, 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 presented in the previous section is applied setting an upper bound on the value of the objective function for the weighted sum problem. For the first triangle this upper bound is defined (see [70]), as follows:

$$\Delta = \lambda_1(y_1(x_{i+1}) - 1) + \lambda_2(y_2(x_i) - 1) \tag{6.21}$$

This upper bound is updated during the recursive procedure by means of the new spanning trees generated. In more 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 the efficient solutions and is used to update the upper bound by means of the formula below (see [69]):

$$\Delta = \max\{\lambda_1(y_1(x_{i,j+1}) - 1) + \lambda_2(y_2(x_{i,j}) - 1),\ \ j = 0, ..., r\} \tag{6.22}$$

The algorithm terminates when all the triangles are explored by means of this recursive procedure. The output of the algorithm is a complete set of efficient solutions. The pseudocode of the whole algorithm is shown below.

INPUT: Network (G, c) with $c = (c^1, c^2)$, list of extreme efficient solutions

$x^1, ..., x^s$.

Two_phase_btree

1: $i = 1$
2: $\mathcal{E} = \emptyset$
3: **while** $(i < s)$ **do**
4: $\mathcal{E}_i = \{x^i, x^{i+1}\}$
5: Compute $\lambda_1 = y_2(x^i) - y_2(x^{i+1})$, $\lambda_2 = y_1(x^{i+1}) - y_1(x^i)$ and $c_\lambda = \lambda_1 c^1 + \lambda_2 c^2$
6: **if** $\mathcal{E} \neq \emptyset$ **then**
7: **for** $x^b \in \mathcal{E}$ **do**
8: **if** $y(x^b) \in T_i$, it is not dominated and not equivalent to any $x \in \mathcal{E}_i$ **then**
9: Insert x^b in $\mathcal{E}_i$ and remove x^b from $\mathcal{E}$
10: **end if**
11: **end for**
12: $\Delta = \max\{\lambda_1(y_1(x^{i,j+1}) - 1) + \lambda_2(y_2(x^{i,j}) - 1), j = 0, ..., r\}$, $r + 1 =\mid \mathcal{E}_i \mid$
13: **end if**
14: **else**
15: $\Delta = \mu_\lambda = \lambda_1(y_1(x^{i+1}) - 1) + \lambda_2(y_2(x^i) - 1)$ /* initial value of Delta*/
16: $\mathcal{E} =$ **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 in absolute value associated with x^{i+1} (sorted from the smallest to the biggest)*/
17: **for** $x^b \in \mathcal{E}$ **do**
18: **if** $y(x^b) \in T_i$, it is not dominated and it is not equivalent to any $x \in \mathcal{E}_i$ **then**
19: Insert x^b in $\mathcal{E}_i$ and remove x^b from $\mathcal{E}$
20: **end if**
21: **end for**
22: $i = i + 1$
23: **end while**
24: **for** $\mathcal{E}_i$ $i = 1, ...s - 1$ **do**
25: Remove potential non-efficient solutions
26: **end for**

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

Algorithm 9: Two-phase algorithm for the bi-objective minimum spanning tree problem.

Theorem 23. *The set $\mathcal{E}^* = \cup_{i=1,...,s-1}\mathcal{E}_i$ generated by the algorithm **Two_phase_btree** is a complete set of efficient solutions of the bi-objective minimum spanning tree problem.*

The proof is identical to the proof of Theorem 20 but in this case the solutions are spanning trees and not flows. This is possible as the strategy used for generating a complete set of efficient solutions is independent from the specific network flow problem under consideration, confirming that the approach has a general applicability.

6.4.3 Illustrative example

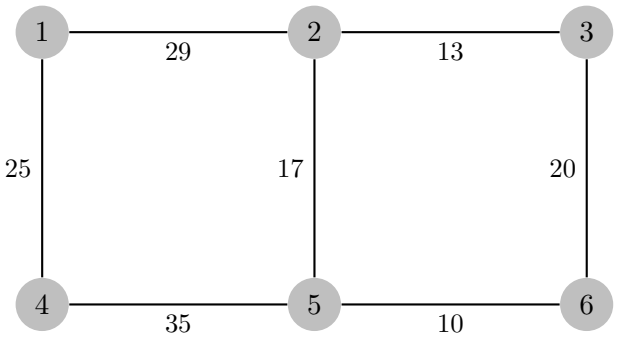
In the following example it is shown the operating principle of the recursive function for generating all the spanning trees of an undirected graph of 6 nodes and 7 edges. Starting from the minimum spanning tree, the recursive procedure is applied.

Data

$$G = (V, E)$$

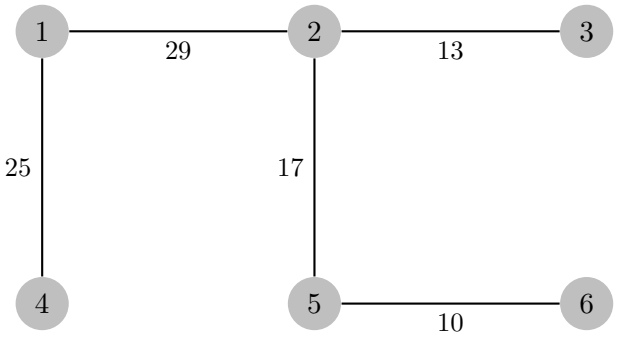
$$V = \{1, 2, 3, 4, 5, 6\}$$

$$E = \{(1, 2), (1, 4), (2, 3), (2, 5), (3, 6), (4, 5), (5, 6)\}$$



$$c = [29, 25, 13, 17, 20, 35, 10]$$

Initial optimal solution T^* with objective value equal to 94



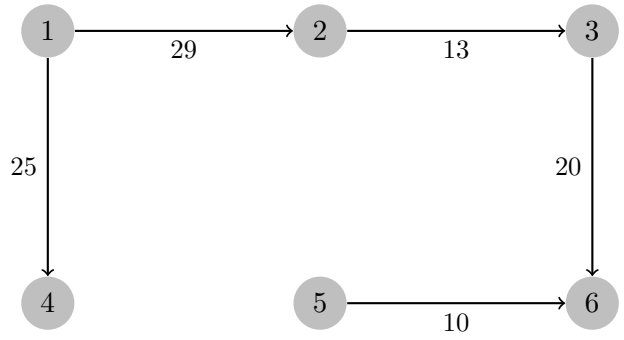
Optimal solution and Reduced costs

$$T^* = \{(1,2),(1,4),(2,3),(2,5),(5,6)\} \quad rc^* = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ -20 \\ -36 \\ 0 \end{bmatrix}$$

Procedure applied on the example

- The edge with minimum non-zero reduced cost (in absolute value) is $(3,6)$.
- Add this edge to the spanning tree T^*.
- The cycle C^* consists in the edges $(2,5)$, $(2,3)$, $(5,6)$ and $(3,6)$.
- Remove from T^* the edge with maximum cost in the cycle C^*, that is, the edge $(2,5)$.

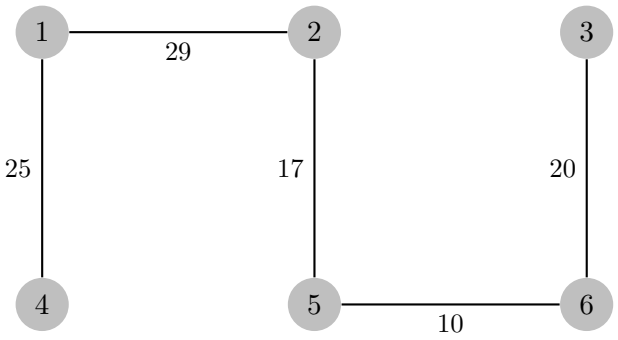
Second best spanning tree with objective value equal to 97



- The edge with minimum non-zero reduced cost (in absolute value) is $(3,6)$.
- Add this edge to the spanning tree T^*.
- The cycle C^* consists in the edges $(2,5)$, $(2,3)$, $(5,6)$ and $(3,6)$.
- Remove from T^* the edge with second maximum cost in the cycle C^*, that is, the edge $(2,3)$.

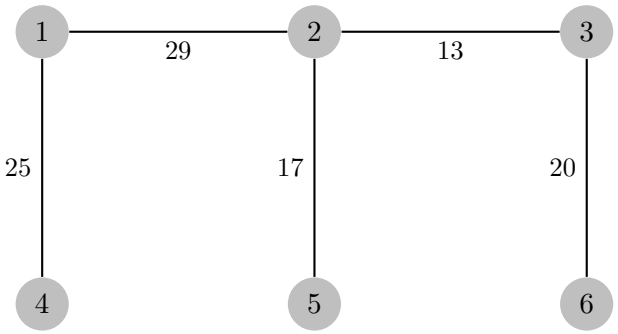
Spanning tree with objective value equal to 100

- The edge with minimum non-zero reduced cost (in absolute value) is $(3,6)$.



- Add this edge to the spanning tree T^*.
- The cycle C^* consists in the edges $(2,5)$, $(2,3)$, $(5,6)$ and $(3,6)$.
- Remove from T^* the edge with third maximum cost in the cycle C^*, that is, the edge $(5,6)$.

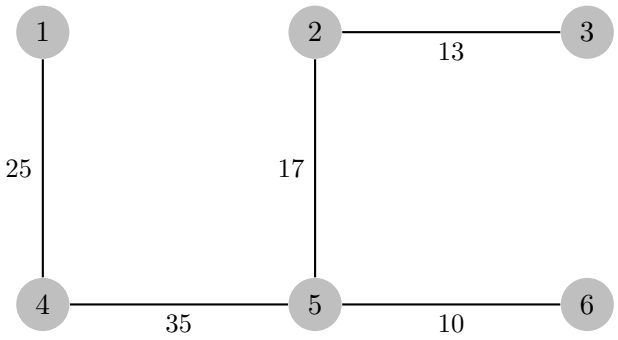
Spanning tree with objective value equal to 103



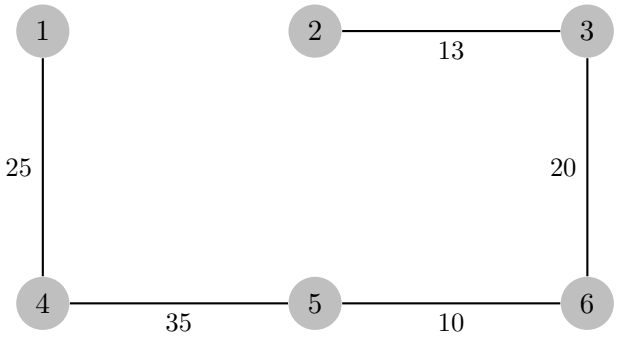
- The edge with second minimum non-zero reduced cost (in absolute value) is $(4,5)$.
- Add this edge to the spanning tree T^*.
- The cycle C^* consists in the edges $(1,2)$, $(1,4)$, $(2,5)$ and $(4,5)$.
- Remove from T^* the edge with maximum cost in the cycle C^*, that is, the edge $(1,2)$.

Spanning tree with objective value equal to 100 ($T^{'}$)

- The edge with minimum non-zero reduced cost (in absolute value) is $(3,6)$.
- Add this edge to the spanning tree $T^{'}$.
- The cycle $C^{'}$ consists in the edges $(2,5)$, $(2,3)$, $(5,6)$ and $(3,6)$.
- Remove from $T^{'}$ the edge with maximum cost in the cycle $C^{'}$, that is, the edge $(2,5)$.

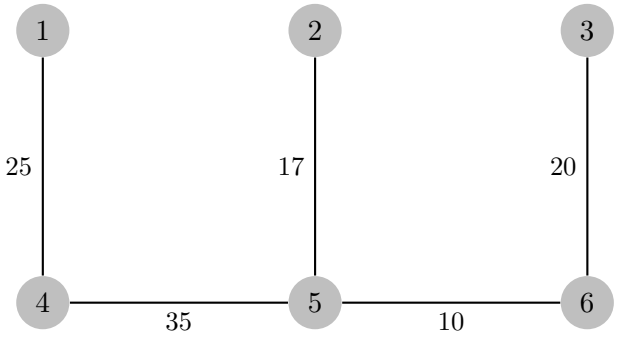


Spanning tree with objective value equal to 103



- The edge with minimum non-zero reduced cost (in absolute value) is $(3, 6)$.
- Add this edge to the spanning tree $T^{'}$.
- The cycle $C^{'}$ consists in the edges $(2, 5)$, $(2, 3)$, $(5, 6)$ and $(3, 6)$.
- Remove from $T^{'}$ the edge with second maximum cost in the cycle $C^{'}$, that is, the edge $(2, 3)$.

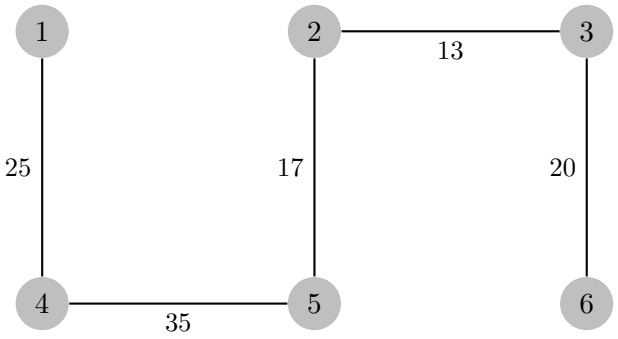
Spanning tree with objective value equal to 107



- The edge with minimum non-zero reduced cost (in absolute value) is $(3, 6)$.

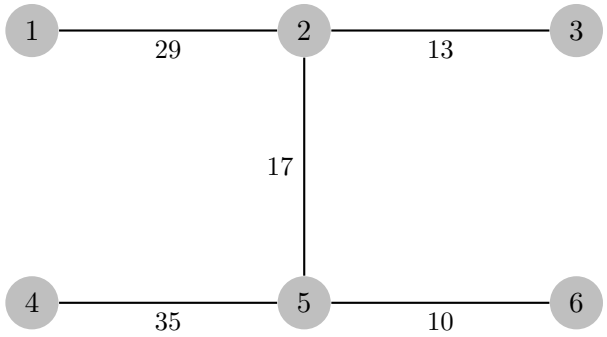
– Add this edge to the spanning tree $T^{'}$.
– The cycle $C^{'}$ consists in the edges $(2,5)$, $(2,3)$, $(5,6)$ and $(3,6)$.
– Remove from $T^{'}$ the edge with third maximum cost in the cycle $C^{'}$, that is, the edge $(5,6)$.

Spanning tree with objective value equal to 110



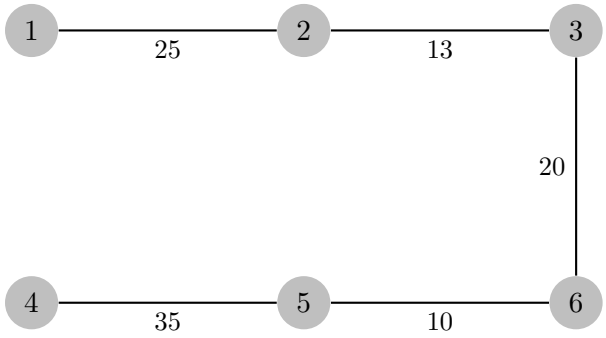
– The edge with second minimum non-zero reduced cost (in absolute value) is $(4,5)$.
– Add this edge to the spanning tree T^{*}.
– The cycle C^{*} consists in the edges $(1,2)$, $(1,4)$, $(2,5)$ and $(4,5)$
– Remove from T^{*} the edge with second cost in the cycle C^{*}, that is, the edge $(1,4)$.

Spanning tree with objective value equal to 104 ($T^{''}$)



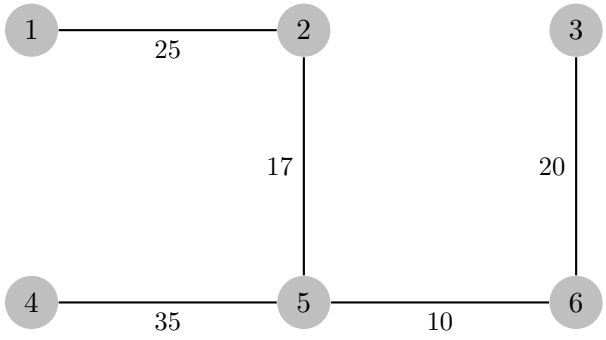
– The edge with minimum non-zero reduced cost (in absolute value) is $(3,6)$.
– Add this edge to the spanning tree $T^{''}$.
– The cycle $C^{''}$ consists in the edges $(2,5)$, $(2,3)$, $(5,6)$ and $(3,6)$.
– Remove from $T^{''}$ the edge with maximum cost in the cycle $C^{''}$, that is, the edge $(2,5)$.

Spanning tree with objective value equal to 107



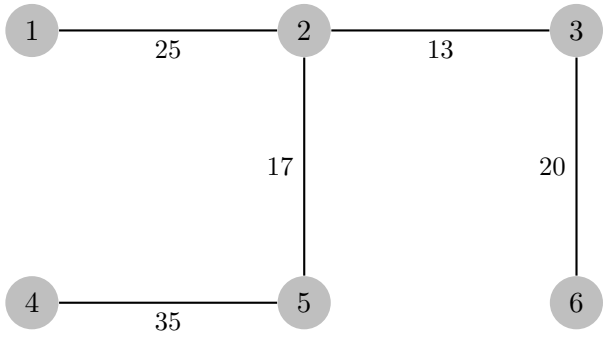
- The edge with minimum non-zero reduced cost (in absolute value) is $(3, 6)$.
- Add this edge to the spanning tree T''.
- The cycle C'' consists in the edges $(2, 5)$, $(2, 3)$, $(5, 6)$ and $(3, 6)$.
- Remove from T'' the edge with second maximum cost in the cycle C'', that is, the edge $(2, 3)$.

Spanning tree with objective value equal to 111



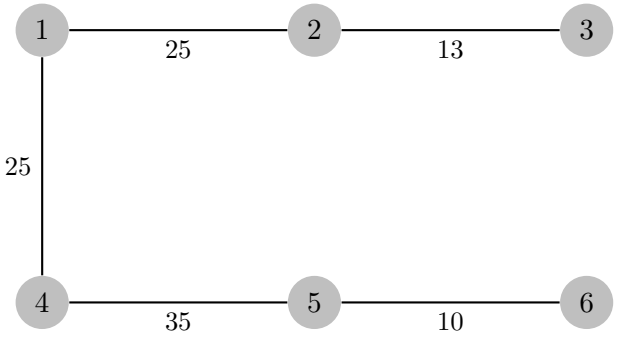
- The edge with minimum non-zero reduced cost (in absolute value) is $(3, 6)$.
- Add this edge to the spanning tree T''.
- The cycle C'' consists in the edges $(2, 5)$, $(2, 3)$, $(5, 6)$ and $(3, 6)$.
- Remove from T'' the edge with third maximum cost in the cycle C'', that is, the edge $(5, 6)$.

Spanning tree with objective value equal to 114



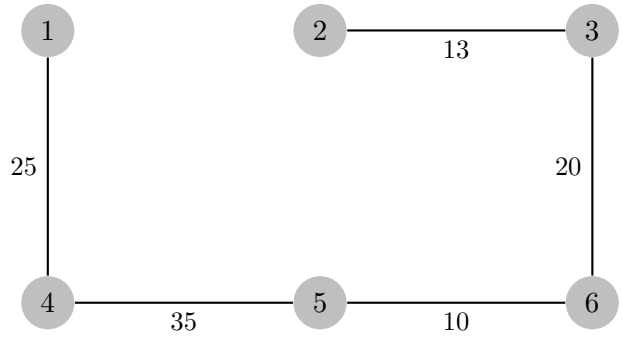
- The edge with second minimum non-zero reduced cost (in absolute value) is $(4, 5)$.
- Add this edge to the spanning tree T^*.
- The cycle C^* consists in the edges $(1, 2)$, $(1, 4)$, $(2, 5)$ and $(4, 5)$
- Remove from T^* the edge with third maximum cost in the cycle C^*, that is, the edge $(2, 5)$.

Spanning tree with objective value equal to 112 (T''')



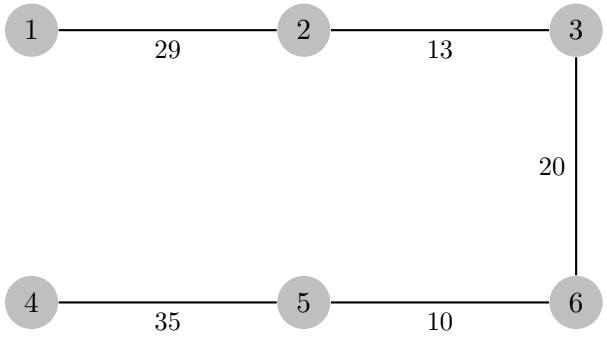
- The edge with minimum non-zero reduced cost (in absolute value) is $(3, 6)$.
- Add this edge to the spanning tree T'''.
- The cycle C''' consists in the edges $(1, 2)$, $(1, 4)$, $(2, 3)$, $(4, 5)$, $(5, 6)$ and $(3, 6)$
- Remove from T''' the edge with maximum cost in the cycle C''', that is, the edge $(1, 2)$.

Spanning tree with objective value equal to 103



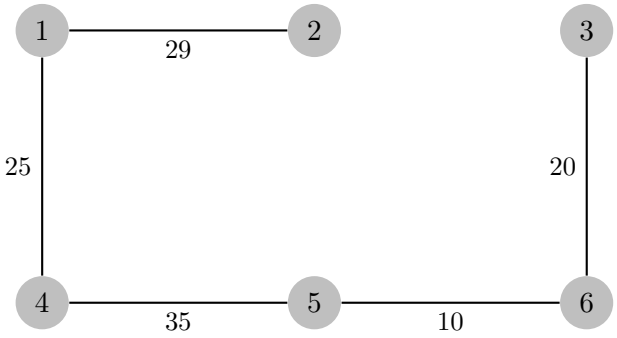
- The arc with minimum non-zero reduced cost (in absolute value) is $(3, 6)$.
- Add this edge to the spanning tree T'''.
- The cycle C''' consists in the edges $(1, 2)$, $(1, 4)$, $(2, 3)$, $(4, 5)$, $(5, 6)$ and $(3, 6)$
- Remove from T''' the edge with second maximum cost in the cycle C''', that is, the edge $(1, 4)$.

Spanning tree with objective value equal to 107



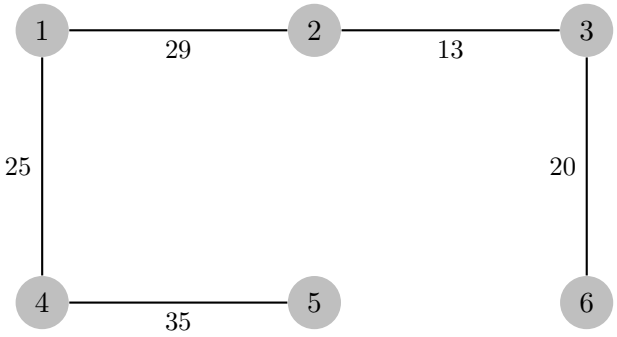
- The edge with minimum non-zero reduced cost(in absolute value) is $(3, 6)$.
- Add this edge to the spanning tree T'''.
- The cycle C''' consists in the edges $(1, 2)$, $(1, 4)$, $(2, 3)$, $(4, 5)$, $(5, 6)$ and $(3, 6)$
- Remove from T''' the edge with third maximum cost in the cycle C''', that is, the edge $(2, 3)$.

Spanning tree with objective value equal to 119



- The edge with minimum non-zero reduced cost (in absolute value) is $(3, 6)$.
- Add this edge to the spanning tree $T^{'''}$.
- The cycle $C^{'''}$ consists in the edges $(1, 2)$, $(1, 4)$, $(2, 3)$, $(4, 5)$, $(5, 6)$ and $(3, 6)$
- Remove from $T^{'''}$ the edge with firth maximum cost in the cycle $C^{'''}$, that is, the edge $(5, 6)$.

Spanning tree with objective value equal to 122



As shown in this example, it is possible to generate, by means of the recursive procedure, all the spanning trees of a graph. As already mentioned, this algorithm can be used in the context of a two-phase method in order to generate a complete set of efficient solutions of a bi-objective minimum spanning tree problem. Indeed, independently from the presence of one or more than one objective, keeping the same topology, the feasible solutions, that is the spanning trees of the graph, will be the same.

For showing how the adaptation of such a procedure works in the two-phase framework, let assume that the following two different costs are associated with the arcs of the graph:

$$c1 = [29, 25, 13, 17, 20, 35, 10],$$
$$c2 = [15, 25, 20, 17, 13, 10, 35].$$

In Table 6.1 all points corresponding to feasible solutions are listed. The two-phase algorithm finds all non-dominated points. In this example we tested both strategies (the weighted sum method on the flow formulation and the dual variant of Benson's algorithm on the Kipp-Martin formulation) for the first phase.

Table 6.1. Feasible points

Dominated		**Non-dominated**		**Type**
Obj1	Obj2	Obj1	Obj2	
		94	112	Supported
		97	108	Non-Supported
		100	107	Non-Supported
		101	105	Non-Supported
		103	103	Non-Supported
104	97			
		104	90	Supported
107	100			
107	93			
		110	85	Non-Supported
111	90			
112	105			
		114	75	Supported
119	98			
122	83			

As shown in Table 6.1, this example is characterized by 8 non-dominated points, 3 of them are supported non-dominated points, the remaining 5 have been found by means of the recursive algorithm applied in the second phase. The algorithm has been implemented in C++ by means of Xcode as for the bi-objective min cost flow problem. The supported ones have been generated by the first phase, implemented with both of the two strategies proposed: the dual variant of Benson's algorithm (by means of Bensolve, [80]) applied on the Kipp-Martin formulation and the weighted sum method applied on the flow formulation solved, varying the weights, with Gurobi solver (see [66]). In order to see how the non-dominated points are positioned in the objective space, Figure 6.1 represents the Pareto frontier.

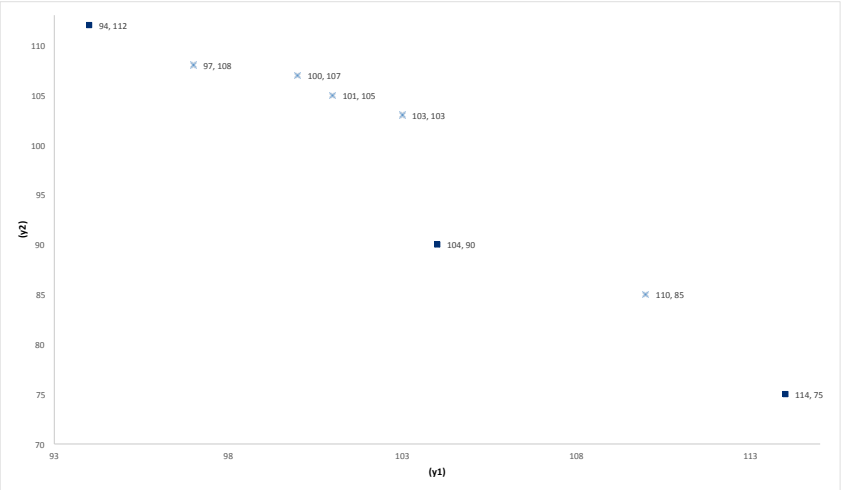


Figure 6.1. Pareto Frontier for the example

We can observe that the three supported non-dominated points identify two triangles. In the first one four non-supported non-dominated points are located. The second one contains the remaining non-supported non-dominated point. Moreover, we can see that some non-supported non-dominated points are far from the boundary of $conv(Y)$. The resolution of the numerical test represented by this example give an experimental confirmation of the correctness of the algorithm proposed, that, with both methodologies for the first phase, is able to generate a complete set of efficient solutions for the bi-objective minimum spanning tree problem. Furthermore, the possibility to use different strategies for the first phase, it will allow us to select the one more suitable to specific problem instances we will face in the future.

Chapter 7

Conclusions and Further Research

As seen in Chapter 2, the new trend in optimization problems on telecommunication networks requires to deal simultaneously with multiple objectives. After proposing, implementing, and testing new models on real and realistic scenarios, it appeared with evidence the necessity to have more powerful tools for multicriteria integer optimization problems on networks. Hence, for this reason, after an intensive study of the literature in the area, the thesis focused on the possibility of designing a quite general method to cope with bi-objective integer optimization problems on networks. The procedure described in Chapter 4 seems to meet this requirement of generality. To verify if the approach was applicable we chose two classical bi-objective integer network optimization problems: the integer min cost flow and the minimum spanning tree problem. The general scheme adapts very well to both problems and can be easily implemented. For the integer min cost flow problem, the numerical tests performed on a selection of test instances, taken from the literature, permit to verify that the algorithm finds a complete set of efficient solutions. For the minimum spanning tree problem, a numerical example, automatically generated, using two alternative methods for the first phase, confirm the practicability of the approach. Further research will be devoted to improving the efficiency of the implementation of the algorithm, also utilizing Bensolve in the second phase in order to speed up the whole procedure, as mentioned in Chapter 5. Moreover, an extensive computational study will be performed for both problems, considering also larger and different instances.

Bibliography

[1] B. Addis and al. Energy management through optimized routing and device powering for greener communication networks. *IEEE/ACM Transactions on Networking (ToN)*, Vol. 22:313–325, 2014.

[2] A. Ahmad and al. Power-aware logical topology design heuristics in wavelength-routing networks. *Proc. ONDM, Bologna, Italy*, 2011.

[3] R. Ahuja, T. Magnanti, and J. Orlin. *Network Flows.Theory, Algorithms and Applications*. Prentice Hall, 1993.

[4] A. Aldraho and A. A. Kist. Enabling energy efficient and resilient networks using dynamic topologies. *Proc. SustainIT, Pisa, Italy*, pages 525–530, 2012.

[5] L. Amorosi and al. Optimal sustainable management of backbone networks. *IEEE, International Conference on Transparent Optical Networks*, pages 1–4, 2016.

[6] L. Amorosi and M. Ehrgott. A new two-phase algorithm for the bi-objective integer min cost flow problem. *Department of Statistical Sciences, work in progress*, 2017.

[7] L. Amorosi and J. Puerto. Two-phase strategies for the bi-objective minimum spanning tree problem. *Department of Statistical Sciences, work in progress*, 2017.

[8] J. Andrews and al. What will 5g be? *IEEE Journal on Selected Areas in Communications*, Vol. 32:1065–1082, 2014.

[9] D. Applegate, R. Bixby, V. Chvatal, and W. Cook. Tsp cuts which do not conform to the template paradigm. *Computational Combinatorial Optimization*, pages 261–303, 2001.

[10] L. Atzori, A. Iera, and G. Morabito. The internet of things: A survey. *Computer networks*, Vol. 54:2787–2805, 2010.

[11] 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. 13:1–24, 1998.

[12] T. Bertsimas and R. Freund. *Data, Models and Decisions: The Fundamental of Management Science*. South-Western College, 2000.

[13] R. Bolla, R. Brischi, F. Davoli, and F. Cucchietti. Energy efficiency in the future internet: a survey of existing approaches and trends in energy-aware fixed network infrastructures. *IEEE Communications Surveys & Tutorials*, Vol. 13:223–244, 2011.

[14] H. Calvete and M. Mateo. A sequential network-based approach for the multiobjective network flow problem with preemptive priorities. *Multi-objective programming and goal programming: theory and applications, Lecture Notes in Economics and Mathematical Systems*, Vol. 432:74–86, 1996.

[15] M. Cardei and D. Du. Improving wireless sensor network lifetime through power aware organization. *Wireless Networks*, Vol. 11:333–340, 2005.

[16] C. Cavdar and al. Design of green optical networks with signal quality guarantee. *Proc. ICC, Ottawa, Canada*, 2012.

[17] J. Chabarek and al. Power awareness in network design and routing. *Proc. of IEEE INFOCOM, Phoenix, USA*, 2008.

[18] V. Chankong and Y. Y. Haimes. *Multiobjective Decision Making: Theory and Methodology*. Courier Dover Publications, 1983.

[19] L. Chiaraviglio and al. Increasing device lifetime in backbone networks with sleep modes. *Proc. of the SoftCOM, Primosten, Croatia*, 2013.

[20] L. Chiaraviglio and al. Is green networking beneficial in terms of device lifetime? *IEEE Communications Magazine*, Vol. 53:232–240, 2015.

[21] L. Chiaraviglio, L. Amorosi, and al. Lifetel: Managing the energy-lifetime trade-off in telecommunication networks. *IEEE Communications Magazine*, Vol. 54:150–157, 2016.

[22] L. Chiaraviglio, L. Amorosi, and al. Lifetime-aware ISP networks: Optimal formulation and solutions. *IEEE/ACM Transactions on Networking (ToN)*, 25(3):1924–1937, 2017.

[23] L. Chiaraviglio, L. Amorosi, and al. Optimal superfluid management of 5g networks. *Proc. of 3rd IEEE Conference on Network Softwarization (IEEE NetSoft)*, 2017.

[24] L. Chiaraviglio, M. Mellia, and F. Neri. Minimizing isp network energy cost: Formulation and solutions. *IEEE/ACM Transactions on Networking (TON)*, Vol. 20:463–476, 2012.

[25] A. Coiro, M. Listanti, A. Valenti, and F. Matera. Energy-aware traffic engineering: a routing-based distributed solution for connection-oriented ip networks. *Computer Networks*, Vol. 57:2004–2020, 2013.

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

[27] G. Dahl, K. Jörnsten, and A. Lokketangen. A tabu search approach to the channel minimization problem. *ICOTA95, Chengdu, China*, 1995.

[28] M. Dharmaweera, R. Parthiban, and Y. Sekercioglu. Toward a power-efficient backbone network: The state of research. *Communications Surveys & Tutorials, IEEE*, Vol. 17:198–227, 2014.

[29] J. Diaz. Solving multiobjective transportation problems. *Ekonomicko Mathematicky Obzor*, Vol. 14:267–274, 1978.

[30] M. Ehrgott. Integer solutions of multicriteria network flow problems. *Investigacao Operacional*, Vol. 19:229–243, 1999.

[31] M. Ehrgott. *Multicriteria optimization.* Springer, 2005.

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

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

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

[35] E. Fernández, M. Pozo, J. Puerto, and A. Scozari. Ordered weighted average optimization in multiobjective spanning tree problems. *European Journal of Operational Research*, 260(3):886–903, 2017.

[36] G. Fettweis. The tactile internet: applications and challenges. *IEEE Vehicular Technology Magazine*, Vol. 9:64–70, 2014.

[37] J. Figueira. On the integer bi-criteria network flow problem: A branch-and-bound approach. Technical report, Cahier du LAMSADE, Université Paris-Dauphine, 2002.

[38] F. Francois, N. Wang, K. Moessner, and S. Georgoulas. Optimizing link sleeping reconfigurations in isp networks with off-peak time failure protection. *IEEE TNSM*, Vol. 10:176–188, 2013.

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

[40] X. Gandibleux, D. Vancoppenolle, and D. Tuyttens. A first making use of grasp for solving moco problems. *Technical report,University of Valenciennes, France*, 1998.

[41] A. M. Geoffrion. Proper efficiency and the theory of vector maximization. *Journal of Mathematical Analysis and Applications*, Vol. 22:618–630, 1968.

[42] F. Giroire, D. Mazauric, J. Moulierac, and B. Onfroy. Minimizing routing energy consumption: from theoretical to practical results. *Proc. of IEEE GreenCom, Hangzhou, China*, 2010.

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

[44] M. Gupta and S. Sigh. Greening of the internet. *Proc. of the SIGCOMM, Karlsruhe, Germany*, 2003.

[45] J. Gutiérrez, J. Puerto, and J. Sicilia. The multiscenario lot size problem with concave costs. *European Journal Of Operational Research*, 156(1):162–182, 2004.

[46] H. Hamacher. A note on k best network flows. *Annals of Operations Research*, Vol. 57:65–72, 1995.

[47] H. Hamacher, C. Pdersen, and S. Ruzika. Multiple objective minimum cost flow problems: A review. *European Journal of Operational Research*, 176:1404–1422, 2007.

[48] H. Hamacher and G. Ruhe. On spanning tree problems with multiple objectives. *Annals of Operations Research*, Vol. 52:209–230, 1994.

[49] A. Hamel, A. Löhne, and B. Rudloff. Benson type algorithms for linear vector optimization and applications. *Journal of Global Optimization*, Vol. 59:811–836, 2014.

[50] F. Heyde and A. Löhne. Geometric duality in multiple objective linear programming. *Journal of Optimization*, Vol. 12:836–845, 2008.

[51] F. Huarng, P. Pulat, and A. Ravindran. An algorithm for bicriteria integer network flow problem. *10th International Conference on Multiple Criteria Decision Making, Taipei (Taiwan)*, Vol. 3:305–318, 1992.

[52] F. Idzikowski and al. Dynamic routing at different layers in IP-over-WDM networks maximizing energy savings. *Optical Switching and Networking*, Vol. 8:181–200, 2011.

[53] F. Idzikowski and al. Trend d3.3 final report for the ira energy-efficient use of network core resources. *available on line at: http://www.fp7-trend.eu/system/files/private/71-wp3/d33-final-report-ira.pdf*, 2012.

[54] F. Idzikowski and al. Green horizon: Looking at backbone networks in 2020 from the perspective of network operators. *Proc. of IEEE ICC, Budapest, Hungary*, 2013.

[55] H. Isermann. Proper efficiency and the linear vector maximum problem. *Operations Research*, Vol. 22:189–191, 1974.

[56] R. Jain and S. Paul. Network virtualization and software defined networking for cloud computing: a survey. *Communications Magazine, IEEE*, Vol. 51:24–31, 2013.

[57] F. Kerasiotis and al. Battery lifetime prediction model for a wsn platform. *Fourth International Conference on Sensor Technologies and Applications (SENSORCOMM)*, pages 525–530, 2010.

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

[59] D. Klingam, A. Napier, and J. Stutz. Netgen: a program for generating large scale assignment, transportation, and minimum cost flow problems. *Management Science*, Vol. 20:814–821, 1974.

[60] J. D. Knowles and D. W. Corne. Enumeration of pareto optimal multicriteria spanning trees - a proof of the incorrectness of zhou and gen's proposed algorithm. *European Journal of Operational Research*, Vol. 143:543–547, 2002.

[61] C. Lee and J. Rhee. Traffic grooming for ip-over-wdm networks: Energy and delay perspectives. *IEEE/OSA JOCN*, Vol. 6:96–103, 2014.

[62] H. Lee and P. Pulat. Bicriteria network flow problems: Integer case. *European Journal of Operational Research*, Vol. 66:148–157, 1993.

[63] G. Lin, S. Soh, and K. Chin. Energy-aware traffic engineering with reliability constraint. *Elsevier ComCom*, Vol. 57:115–128, 2015.

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

[65] D. Narsingh. *Graph Theory, with Applications to Engineering & Computer Science*. Dover Publications, INC, 2016.

[66] G. Optimization. *Gurobi Optimizer Reference Manual*. http://www.gurobi.com/documentation, 2017.

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

[68] A. Przybylski, G. X., and M. Ehrgott. The biobjective integer minimum cost flow problem–incorrectness of Sedeño-Noda and González-Martín's algorithm. *Computers and Operations Research*, Vol. 33:1459–1463, 2006.

[69] A. Raith and M. Ehrgott. A comparison of solution strategies for biobjective shortest path problems. *Computers & Operations Research*, Vol. 36:1299–1331, 2009.

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

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

[72] R. Rockafellar. *Convex Analysis*. Princeton University Press, 1972.

[73] A. Sedeño Noda and C. González-Martín. An algorithm for the biobjective integer minimum cost flow problem. *Asia-Pacific Journal of Operational Research*, Vol. 20:241–260, 2003.

[74] 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. 294:222–232, 1987.

[75] P. Serafini. A simulated annealing for multiobjective optimization problems. *10th International Conference on Multiple Criteria Decision Making, Taipei (Taiwan)*, Vol. 1:87–96, 1992.

[76] F. Sourd and O. Spanjaard. A multiobjective branch-and-bound framework: Application to the biobjective spanning tree problem. *INFORMS Journal on Computing*, 2008.

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

[78] E. Ulungu and J. Teghem. The two phases method: an efficient procedure to solve bi-objective combinatorial optimization problems. *Foundations of Computing and Decision Sciences*, Vol. 20:149–165, 1995.

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

[80] B. Weißing and A. Löhne. The vector linear program solver bensolve – notes on theoretical background. *European Journal of Operational Research*, Vol. 260:807–813, 2017.

[81] P. Wiatr, P. Monti, and L. Wosinska. Energy efficiency and reliability tradeoff in optical core networks. *Proc. of OSA OFC, San Francisco, USA*, 2014.

[82] P. Wiatr, P. Monti, and L. Wosinska. Energy efficiency versus reliability performance in optical backbone networks. *Journal of Optical Communications and Networking*, Vol. 7:A482–A491, 2015.

[83] Y. Yang, D. Wang, M. Xu, and S. Li. Hop-by-hop computing for greeninternet routing. *Proc. ICNP, Göttingen, Germany*, 2013.

[84] S. Zhang, D. Shen, and C. Chan. Energy-efficient traffic grooming in wdm networks with scheduled time traffic. *IEEE/OSA JLT*, Vol. 29:2577–2584, 2011.

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