Université libre de Bruxelles
École polytechnique de Bruxelles

Inverse Multi-Objective Combinatorial Optimization

par

Julien Roland

Dissertation originale présentée en vue de l'obtention du diplôme de Docteur en Sciences de l'Ingénieur

Promoteurs: Prof. Yves De Smet
Prof. José Rui Figueira

Année académique 2013 – 2014

Cette thèse de doctorat n'aurait pas pu aboutir sans le soutien, l'aide et le travail de nombreuses personnes.

Tout d'abord, j'aimerais remercier mes directeurs de thèse, les Professeurs José Rui Figueira et Yves De Smet. Je suis très reconnaissant envers José Figueira pour m'avoir proposé de faire une thèse de doctorat lors de mon stage de recherche à l'Instituto Superior Técnico de Lisbonne. Je le remercie également pour son encadrement qui fut, même à longue distance, très important, ainsi que pour son accueil lors de mes visites à Lisbonne et Nancy. Je suis également très reconnaissant envers Yves De Smet pour son encadrement qui fut des plus important jusqu'aux dernières semaines de rédaction, ainsi que pour m'avoir laissé beaucoup de liberté dans mon organisation.

Cette thèse a bénéficié de la contribution de nombreuses personnes. En particulier, j'aimerais remercier les Professeurs Bernard Fortz et Dmitry Podkopaev pour leurs nombreuses remarques, commentaires et idées.

Je tiens à remercier les Professeurs Mauro Birattari, Marc Pirlot, Daniel Vanderpooten, et Philippe Vincke pour avoir accepté de faire partie du jury chargé d'évaluer ce travail. De plus, je remercie les Professeurs Mauro Birattari, Bertrand Mareschal, Marc Pirlot, et Philippe Vincke pour avoir fait partie du comité d'accompagnement de ma thèse de doctorat.

Au cours de la rédaction de ce travail, j'ai reçu l'aide de nombreuses personnes. En particulier, j'aimerais remercier Marc Pirlot, Karim Lidouh, et Stefan Eppe. Ils m'ont tous aidé à améliorer de manière significative la qualité de ce travail.

D'un point de vue plus personnel, je souhaite remercier ma famille et tout particulirement mes parents pour m'avoir encouragé à poursuivre des études universitaires et pour m'avoir soutenu tout au long de celles-ci. Finalement, je serai toujours reconnaissant envers Mademoiselle Mélanie Bance pour m'avoir encouragé à aller au bout de ce travail et de m'avoir soutenu lorsque je n'y croyais plus.

Contents

List of Tables

List of Figures

List of Notations

$[\alpha \bullet \beta \bullet \gamma]$	an inverse multi-objective optimization problem	63
A	set of arcs	34
$D = (V, A)$	Directed graph (digraph)	34
V	set of vertices	34
BIP	linear binary integer problem	13
C	profit matrix	26
c	profit vector	12
C_i	the i-th row of C	5
$conv.hull$	convex hull	7
$E(X, C)$	set of all the efficient solutions of (X, C)	26
$F(x)$	outcome vector	25
IP	linear integer problem	12
KP	the {0,1}-knapsack problem	13
LP	linear optimization problem	12
MOCO	a multi-objective combinatorial optimization problem	26
MOLP	multi-objective linear optimization problem	26
$ND(X, C)$	set of all the non-dominated outcome vectors of (X, C)	26
(X, C)	instance of a multi-objective combinatorial optimization problem	26
(X, c)	instance of a linear combinatorial optimization problem	12
X	set of all the feasible solutions	12
$x^\top$	the transpose of x	5

Chapter 1

Introduction

An optimization problem consists of finding the best item in a set of feasible solutions. Each item is evaluated according to an objective function, which allows us to rank the items from the best to the worst ones. Hence, an optimization problem is characterized by a pair composed of an objective function and a feasible set (Papadimitriou and Steiglitz 1982).

One of the most well-known optimization problems is the shortest path problem, which consists of finding the shortest route between two distinct nodes in a network. This problem can be used to answer questions such as: "What is the shortest path from *Iéna* station to *Alésia* station in the Paris Metro subway network?". In this case, the feasible set is characterized by all the feasible paths between these two stations and the objective function takes into account the time traveled between stations in order to evaluate the travel time between *Iéna* and *Alésia*.

In this thesis, we focus ourselves on a particular class of optimization problems, the so-called *combinatorial optimization problems*. In this case, the feasible set is a finite collection of objects and its cardinality is frequently exponential in the size of the collection's representation (Schrijver 2003). Typical examples of combinatorial optimization problems are the shortest path problem, the knapsack problem, and the assignment problem.

As illustrated previously, optimization problems can be used to formulate questions from the real-world. Modeling such a problem requires the precise definition of the values of parameters in order to build the objective function and the feasible set. Those values rely on some assumptions as well as on the accuracy of the evaluations. Therefore, a small perturbation of these parameters can lead to another optimization problem that still remains suitable to model the initial question. However, even though these models are very close, they could lead to different optimal solutions.

The study of the link between a small adjustment of the parameters and a change in the optimal solution is at the core of a mathematical programming approach called *inverse optimization*. The most studied question in inverse optimization can be described as the problem of finding a minimal adjustment of the objective function parameters such that a given feasible solution becomes an optimal one (Ahuja and Orlin 2001, Heuberger 2004). Over the last years, this approach has received great attention within the combinatorial optimization community, and has shown its crucial importance in geophysical sciences, transportation and traffic flow, among others (see, for example, Tarantola (1987), Burton and Toint (1992), Sokkalingam et al. (1999), Ahuja and Orlin (2001)).

Let us illustrate inverse optimization in the problem of finding the shortest path from *Iéna* station to *Alésia* station. It is known that the theoretical shortest path is not always the one chosen by users in practice (Burton and Toint 1992). Indeed, usually users choose their route based on a variety of factors, such as the travel time, the safety, the number of transfers from one train to another, or congestion. Inverse optimization can be used to incorporate the routes that are actually used into the model, modifying the *a priori* costs in order to ensure the optimality of these paths.

In real-world applications, it is common to encounter decision making problems that are by their very nature multidimensional, *i.e*, their feasible solutions are evaluated with respect to multiple criteria. This is a prominent aspect in situations modeled as knapsack problems, such as capital budgeting (Bhaskar 1979, Rosenblatt and Sinuany-Stern 1989), planning remediation of contaminated lightstation sites (Jenkins 2002), selecting transportation investments (Teng and Tzeng 1996), and relocating problems arising in conservation biology (Kostreva et al. 1999).

In this context, the mathematical programming model is a *multi-objective optimization* problem. These problems are characterized by a pair composed of a feasible set and a vector of objective functions that are often contradictory (Zeleny 1974, Steuer 1986, Ehrgott 2005). The main difficulty faced with such problems is that there is not only one objective function to be maximized, but a set of functions that has to be maximized “simultaneously”. Since there is generally no *ideal* feasible solution that is simultaneously optimal for all objective functions, the resolution of such a problem usually leads to finding the so-called *efficient solutions*. An efficient solution is characterized by the fact that it is not possible to find another feasible solution that leads to an improvement of the outcomes of all objective functions, without a degradation of the outcome of at least one objective function.

The initial question addressed in this thesis is how to take into account the multi-objective aspect of decision problems in inverse optimization. This extension has been mentioned in Ahuja and Orlin (2001), but to our best knowledge, it has not been covered yet. The most straightforward extension consists of finding a minimal adjustment of the objective functions coefficients such that a given feasible solution becomes efficient. However, there is not only a single question raised by inverse multi-objective optimization, because there is usually not a single efficient solution. The way we define inverse multi-objective optimization takes into account this important aspect. This gives rise to many questions which are identified by a precise notation that highlights a large collection of inverse problems that could be investigated. In this thesis, a selection of inverse problems are presented and solved. This selection is motivated by their possible applications and the interesting theoretical questions they can rise in practice.

In addition, two main fields of application, where inverse multi-objective optimization is of a significant interest, have been identified. In stability analysis, it can be used to assess the stability of a feasible solution to remain efficient when the problem parameters are perturbed. To the best of our knowledge, this approach leads to the first algorithm that allows to compute the stability radius in a reasonable amount of time.

In group decision making, it can be used to compute compromise solutions among different experts. Such a problem requires to select a solution that satisfies a set of constraints and represents a compromise among the group of experts. This decision making task can be modeled as a multi-objective combinatorial optimization problem, where each objective function is an expert's point of view over the feasible set. Let us observe that if there exists an ideal solution, then there is a consensus among the experts. Based on this observation, a compromise solution may be determined by finding a minimal adjustment of the experts' evaluations so that an ideal solution exists.

Contributions

Most of the results presented in this thesis have been published in three international journals and presented in three conferences.

The contributions on the inverse {0,1}-knapsack problem, presented in Chapter 6, have been published in *Discrete Optimization* (Roland et al. 2013a). The first work on inverse multi-objective optimization has been accepted for publication in *Discrete Applied Mathematics* and the corrected proof is available online on the journal's website (Roland et al. 2013b). Finally, the contributions on the stability radius, presented in Chapters 7 and 8, have been published in *4OR – A Quarterly Journal of Operations Research* (Roland et al. 2012).

These results have also been presented in the following three conferences:

- "Inverse combinatorial optimization under the Chebyshev norm and its extensions to multi-objective optimization", ORBEL26, 26th Annual Conference of the Belgian Operations Research Society, Brussels, Belgium, February 2-3, 2012.

- "A propos de l'analyse de stabilité de l'ensemble des solutions efficaces dans les problèmes combinatoires multi-objectifs", 13ème Congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision, Angers, France, April 11 -13, 2012.

- "The Inverse Multi-Objective {0,1}-Knapsack Problem under the Chebyshev Distance", The 21st International Conference on Multiple Criteria Decision Making, Jyvaskyla, Finland, June 13 - 17, 2011.

This thesis incorporates all the results presented in the three articles and the three presentations cited above. However, let us point out that more details are provided and that several proofs have been completely rewritten in order to ease their understanding. Furthermore, inverse multi-objective optimization is presented in a totally new way through a taxonomy of inverse problems.

Outline

This thesis is organized as follows. The following four chapters present the background on which relies this research. In Chapter 2, several important concepts, their definitions and the basic notation are introduced. In Chapter 3, combinatorial optimization is defined, a quick overview of the theory of complexity is given, and various methods for solving such problems are presented and illustrated. In Chapter 4, multi-objective combinatorial optimization is defined, and various methods are presented. In Chapter 5, a quick review of inverse optimization is provided, a method for solving the inverse shortest path problem is presented, and a previously proposed method for solving the inverse $\{0,1\}$-knapsack problem is analyzed and its incorrectness is proved.

The following three last chapters constitute the novelty of this thesis. In Chapter 6, several methods for solving two inverse $\{0,1\}$-knapsack problems are presented. These problems are classified in complexity classes and numerical results from computational experiments are reported. In Chapter 7, inverse optimization is extended to multi-objective optimization, a taxonomy of inverse problems is proposed, and a collection of algorithms along with numerical results from computational experiments are presented. In Chapter 8, inverse multi-objective optimization is applied to stability analysis and group decision making. We conclude this thesis in Chapter 9, with a summary of the main results, several remarks, and potential directions for future research.

Chapter 2

Preliminary notions

This section addresses several important concepts, their definitions and the basic notation required for the remaining chapters of this thesis. First, let us present the classical set notation.

- Let $\mathbb{Z}$ be the set of integers,
- $\mathbb{R}$ the set of reals,
- $\mathbb{R}_+ = \{x \in \mathbb{R} : x \geqslant 0\}$ the set of positive reals including zero,
- $\mathbb{N} = \{x \in \mathbb{Z} : x \geqslant 0\}$ the set of natural numbers, that is the set of positive integers including zero.

Let $J = \{1, 2, \dots, j, \dots, n\}$ and $I = \{1, 2, \dots, i, \dots, q\}$ be two sets of elements indices. Let $\mathbb{R}^n = \{(x_1, x_2, \dots, x_j, \dots, x_n) : x_j \in \mathbb{R} \text{ for } j \in J\}$ be the set of real-valued vectors of length $n \geqslant 1$, $\mathbb{R}^{q\times n} = \{(C_1, C_2, \dots, C_i, \dots, C_q) : C_i \in \mathbb{R}^n \text{ for } i \in I\}$ be the set of real-valued matrices composed of n columns and q rows denoted by C_i with $i \in I$. A vector $x \in \mathbb{R}^n$ is a matrix composed of 1 column and n rows, and the transpose of x, denoted by $x^\top$, is a matrix composed of n columns and 1 row. By abuse of notation, the *zero vector* 0 is a vector with all the components equal to zero. The canonical basis of $\mathbb{R}^n$ is denoted by the n vectors e^j with $j \in J$.

For any vector $x \in \mathbb{R}^n$, and real number $p \geqslant 1$, the L_p norm of x is defined by

$$|x|_p = \Big(\sum_{j\in J} |x_j|^p\Big)^{\frac{1}{p}},$$

where L_1, L_2, and L_∞ are respectively the so-called Manhattan, Euclidean, and Chebyshev norms. For instance, $|x|_1 = \sum_{j\in J} |x_j|$, and $|x|_\infty = \max_{j\in J} |x_j|$.

The distance between two vectors $c, d \in \mathbb{R}^n$ can be computed by the L_p norm of $c - d$.

The distance between two matrices $C, D \in \mathbb{R}^{q\times n}$ can be measured by the norm of $C - D$ (see, for example, Deza and Deza (2009), or Horn and Johnson (1990)). Let $|\cdot|$ denote a vector norm, and $||\cdot||$ a matrix norm. If $C - D$

is treated as a vector of size qn, then vector norms such as the L_p norm can be used. For instance, $|C - D|_1 = \sum_{i \in I} \sum_{j \in J} |C_{ij} - D_{ij}|$, and $|C - D|_\infty = \max_{i \in I, j \in J} |C_{ij} - D_{ij}|$. Otherwise, one may consider matrix norms such as the maximum column sum $||C - D||_1 = \max_{j \in J} \sum_{i \in I} |C_{ij} - D_{ij}|$, or the maximum row sum $||C - D||_\infty = \max_{i \in I} \sum_{j \in J} |C_{ij} - D_{ij}|$. In what follows, it is assumed that $\delta : \mathbb{R}^{q \times n} \times \mathbb{R}^{q \times n} \to \mathbb{R}_+$ is a distance function that can be linearized by a set of linear constraints, *i.e.*, can be replaced by a linear function and a set of constraints in a linear program (*a linearizable function*). L_1 and L_∞ norms are examples of such a function. Their linearizations are detailed in Chapter 6 and Chapter 7.

Let $x, y \in \mathbb{R}^n$ be two vectors, we will note

- $x < y$ iff $\forall j \in J : x_j < y_j$,
- $x \leqq y$ iff $\forall j \in J : x_j \leqslant y_j$,
- $x \neq y$ iff $\exists j \in J : x_j \neq y_j$,
- $x \leq y$ iff $x \leqq y$ and $x \neq y$,

and the binary relations $\geqq$, $\geq$, and $>$ are defined in a similar way.

Let $V, W \subseteq \mathbb{R}^n$ denote two sets of vectors. Then, the set addition of V and W (denoted by $V \oplus W$) can be stated as follows,

$$V \oplus W = \{x \in \mathbb{R}^n : x = x^1 + x^2, x^1 \in V, x^2 \in W\},$$

and by abuse of notation, $\{x\} \oplus W$ is also noted $x \oplus W$.

Definition 1 (Open Hypersphere (Steuer 1986)). *Consider the space $\mathbb{R}^n$ with the euclidean distance denoted by $L_2(x, y) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$, for all $x, y \in \mathbb{R}^n$. A n-dimensiomal open hypersphere centered at $x^* \in \mathbb{R}^n$ with radius $\epsilon > 0$ is the set*

$$H_\epsilon(x^*) = \Big\{x \in \mathbb{R}^n : L_2(x, x^*) < \epsilon\Big\},$$

and by abuse of notation, $H_\epsilon(x^*)$ is also noted H_ϵ.

Definition 2 (Interior Point (Steuer 1986)). *A point $x^* \in S \subset \mathbb{R}^n$ is an interior point of S if and only if x^* belongs to a n-dimentional open hypersphere H_ϵ centered at x^* such that $H_\epsilon \subset S$.*

Definition 3 (Boundary Point (Steuer 1986)). *A boundary point of $S \subset \mathbb{R}^n$ is a point $x^* \in \mathbb{R}^n$ such that every n-dimentional open hypersphere H_ϵ centered at x^* contains points in S and points not in S.*

Definition 4 (Bounded Set (Steuer 1986)). *A set $S \subset \mathbb{R}^n$ is bounded if and only if there exists an n-dimensional hypersphere that contains S. Otherwise, the set is unbounded.*

Consider q vectors $x^1, x^2, \ldots, x^j, \ldots, x^q \in \mathbb{R}^n$, and q scalars λ_1, λ_2,. . .,λ_j,. . ., $\lambda_q \geqslant 0$ with $\sum_{j=1}^{q} \lambda_j = 1$. The expression $\lambda_1 x^1 + \lambda_2 x^2 + \ldots + \lambda_j x^j + \ldots + \lambda_q x^q$ is said to be a convex combination of vectors $x^1, x^2, \ldots, x^j, \ldots, x^q$.

Definition 5 (Convex Set (Steuer 1986))**.** *A set $S \subset \mathbb{R}^n$ is convex if and only if for any $x^1, x^2 \in S$ the point $\lambda x^1 + (1 - \lambda)x^2 \in S$ for all $\lambda \in [0, 1]$. Otherwise, the set is nonconvex.*

Definition 6 (Convex Hull (Padberg 1995, Schrijver 2003))**.** *Let $S \subseteq \mathbb{R}^n$. The set*

$$\Big\{ \sum_{i=1}^{t} \mu_i x^i : t \geqslant 1, \mu \in \mathbb{R}^t_+, \sum_{i=1}^{t} \mu_i = 1, \text{and } x^1, \ldots, x^t \in S \Big\}$$

is the convex hull of S, or conv.hull(S) for short.

Definition 7 (Extreme Point (Steuer 1986))**.** *A point $x^* \in S \subset \mathbb{R}^n$ is an extreme point if and only if two points $x^1, x^2 \in S$, with $x^1 \neq x^2$ do not exist such that $x^* = \lambda x^1 + (1 - \lambda)x^2$ for some $\lambda \in]0, 1[$.*

Definition 8 (Cone (Padberg 1995))**.** *A subset $S \subseteq \mathbb{R}^n$ is a cone if and only if $x^1, x^2 \in S$ implies $\lambda_1 x^1 + \lambda_2 x^2 \in S$, for all $\lambda_1 \geqslant 0$ and $\lambda_2 \geqslant 0$.*

Let us define the following two cones:

$$\mathbb{R}^n_{\geqq} = \Big\{ x \in \mathbb{R}^n : x = \sum_{j \in J} e^j \alpha_j, \ \alpha_j \geqslant 0, \ j \in J \Big\},$$

$$\mathbb{R}^n_{\leqq} = \Big\{ x \in \mathbb{R}^n : x = \sum_{j \in J} e^j \alpha_j, \ \alpha_j \leqslant 0, \ j \in J \Big\}.$$

The graphical representation of these cones are presented in Figures 2.1 and 2.2.

Definition 9 (Displaced Cone)**.** *Let $S \subseteq \mathbb{R}^n$ be a cone and $y \in \mathbb{R}^n$ a vector. The set $y \oplus S$ is a displaced cone, which represents the translation of S from the origin to vector y.*

Let us illustrate the concept of displaced cone with a vector $y \in \mathbb{R}^2$. The Figures 2.3 and 2.4 represent $y \oplus \mathbb{R}^2_{\geqq}$ and $y \oplus \mathbb{R}^2_{\leqq}$, respectively.

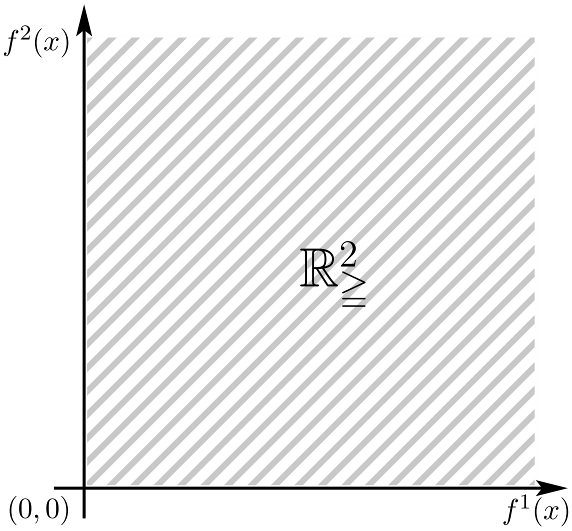

Figure 2.1: The cone $\mathbb{R}^2_{\geqq}$ represented graphically by the hatched zone.

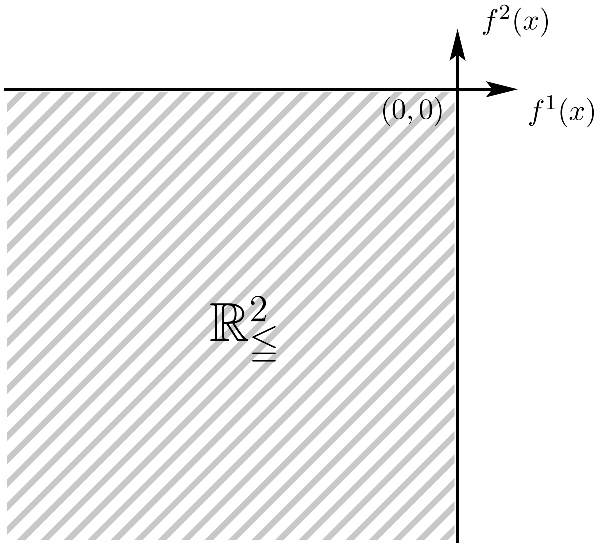

Figure 2.2: The cone $\mathbb{R}^2_{\leqq}$ represented graphically by the hatched zone.

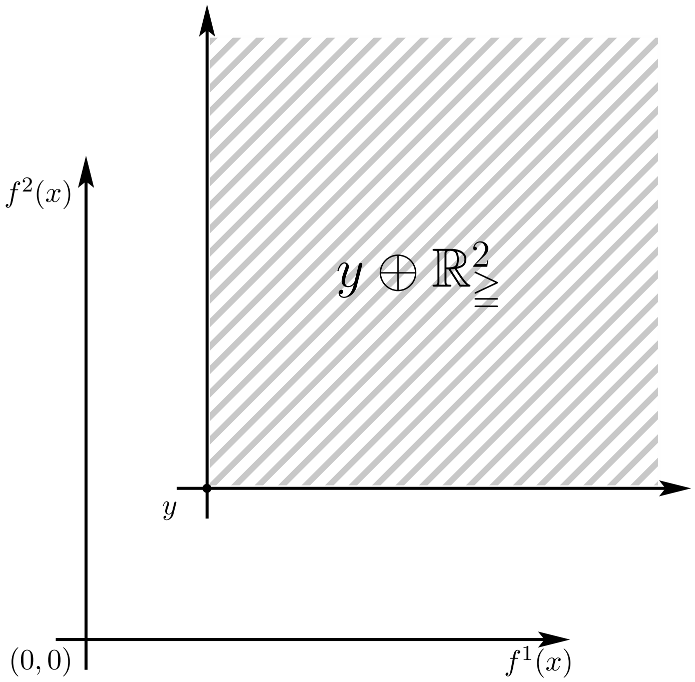

Figure 2.3: The displaced cone $y \oplus \mathbb{R}^2_{\geqq}$ represented graphically by the hatched zone.

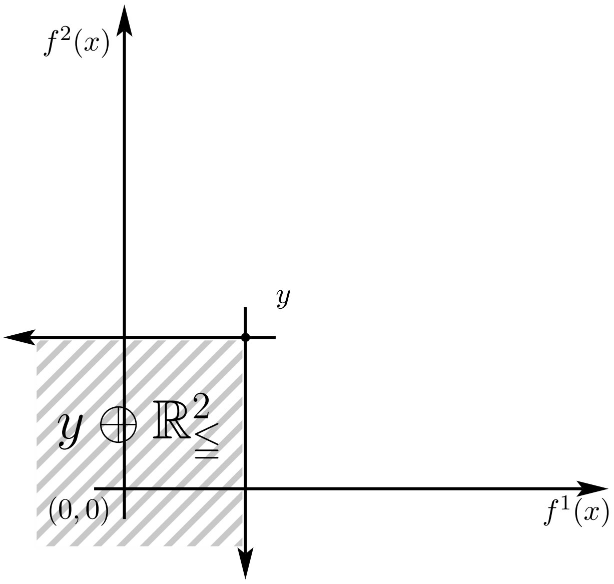

Figure 2.4: The displaced cone $y \oplus \mathbb{R}^2_{\leqq}$ represented graphically by the hatched zone.

Chapter 3

Combinatorial optimization

Combinatorial optimization consists of finding an optimal solution in a finite collection of objects. This class of problem is formally defined and several related problems are also considered. A quick overview of the theory of complexity is given. Finally, various methods are presented and illustrated on the well-known {0,1}-knapsack problem. We will consider only methods that are required for a good understanding of the next chapters.

3.1 Concepts, definitions and notation

Combinatorial optimization is defined by Schrijver (2003) as the problem of finding an optimal object in a finite collection of objects, where the number of objects is typically exponential in the size of the collection's representation. In this thesis, we restrict ourselves to a particular class of combinatorial optimization problems that are defined as follows (Nemhauser and Wolsey 1988).

Definition 10 (An instance of a linear combinatorial optimization problem). *Let* $E = \{e_1, e_2, \ldots, e_j, \ldots, e_n\}$ *denote a finite set,* $X \subseteq 2^E$ *a family of subsets of* E *(the feasible set), and* $f : 2^E \to \mathbb{N}$ *a linear profit function. For each solution* $S \in X$*, consider the expression* $f(S) = \sum_{e \in S} f(\{e\})$*. An instance of a linear combinatorial optimization problem is a pair* (X, f)*.*

Definition 11 (A linear combinatorial optimization problem). *A linear combinatorial optimization problem is characterized by a set* Π *of instances* (X, f)*. The problem consists of finding, for a given instance* (X, f)*, a solution* $S^* \in X$ *such that* $S^* \in \arg\max\{f(S) : S \in X\}$*.*

From Definition 11, we may conclude that any set of instances (*i.e.*, any set of pairs (X, f)) is an optimization problem. However, the instances that constitute an optimization problem usually share a common structure. For example, in the shortest path problem the feasible set X is always characterized by a precise model which ensures that each feasible solution is a path between a source node and a target node.

The objective function $f : 2^E \to \mathbb{N}$ can be characterized by a vector $c \in \mathbb{N}^n$. For this purpose, each subset S of E is completely defined by an incidence vector.

Definition 12 (An incidence vector)**.** *Consider a subset $S \subseteq E$. The incidence vector $x \in \{0,1\}^n$ of E defining S can be stated as follows,*

$$x_j = \begin{cases} 1, & \text{if } e_j \in S, \\ 0, & \text{otherwise.} \end{cases}$$

Consequently, the feasible set is rewritten as $X \subseteq \big\{x : x \in \{0,1\}^n\big\}$. For each feasible solution $x \in X$, the objective function is defined by $f(x) = c^\top x = \sum_{j \in J} c_j x_j$, where $J = \{1, 2, \ldots, j \ldots, n\}$ is the set of element indices, and $c \in \mathbb{N}^n$ is the so-called *profit vector*. Therefore, a linear combinatorial optimization problem is defined by pair (X, c) and consists of finding $x^* \in X$ such that $x^* \in \arg\max\{f(x) : x \in X\}$.

Combinatorial optimization problems are related to linear optimization problems. These problems are very well-known and can be solved by the simplex algorithm or interior point methods (Vanderbei 2008).

Definition 13 (Linear optimization problem (LP))**.** *An instance of a linear optimization problem is defined by* $\max\{c^\top x : Ax \leqq b, x \in \mathbb{R}^n\}$*, where $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, and $b \in \mathbb{R}^m$.*

For some particular problems, the variables (not necessarily all) have to take an integer value, in which case the model is a linear mixed integer problem. When all the variables have to take an integer value the model is a linear integer problem.

Definition 14 (Linear mixed integer problem)**.** *Let $J^{\mathbb{Z}} \subset J$ denotes the set of integer components. An instance of a linear mixed integer optimization problem is defined by* $\max\{c^\top x : Ax \leqq b, x_j \in \mathbb{Z} \text{ for all } j \in J^{\mathbb{Z}}, x_j \in \mathbb{R} \text{ for all } j \notin J^{\mathbb{Z}}\}$*, where $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, and $b \in \mathbb{R}^m$.*

Definition 15 (Linear integer problem (IP))**.** *An instance of a linear integer optimization problem is defined by* $\max\{c^\top x : Ax \leqq b, x \in \mathbb{Z}^n\}$*, where $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, and $b \in \mathbb{R}^m$.*

Every combinatorial optimization problem can be formulated as a binary integer problem that is an IP where all variables have to be either 0 or 1. Indeed, it is easy to transform a combinatorial optimization problem into a BIP by formulating a shortest path problem in a directed graph, where each feasible solution $x \in X$ is a path between the source and the target nodes. This is illustrated in Chapter 5, where the knapsack problem is reduced to the shortest path problem. However, the number of inequalities in the BIP is potentially exponential in the size of E.

Definition 16 (Linear binary integer problem (BIP)). *An instance of a linear binary integer optimization problem is defined by* $\max\{c^\top x : Ax \leqq b, x \in \{0,1\}^n\}$*, where* $c \in \mathbb{R}^n$*,* $A \in \mathbb{R}^{m\times n}$*, and* $b \in \mathbb{R}^m$*.*

The {0,1}-knapsack problem (KP) is a well-known classical linear combinatorial optimization problem (see, for example, Martello and Toth (1990) or Kellerer et al. (1994) for a complete review of knapsack problems, their particular cases, extensions, and formulations). Let $J = \{1, 2, \dots, j, \dots, n\}$ denotes a set composed of n items with profits $c_j \in \mathbb{N}$ and weights $w_j \in \mathbb{N}$, for each $j \in J$. Let $W \in \mathbb{N}$ denote the capacity of a knapsack. The {0,1}-knapsack problem consists of selecting a subset $S \subseteq J$, such that the sum of the profits of the elements of S is maximized and the sum of weights of the same elements does not exceed the capacity of the knapsack.

The problem can be formulated as a BIP as follows.

$$\begin{aligned} \max \quad f(x) = & \sum_{j\in J} c_j x_j & \\ \text{subject to:} \quad & \sum_{j\in J} w_j x_j & \leqslant W \\ & x_j & \in \{0,1\},\ j \in J. \end{aligned} \tag{KP}$$

Consequently, an instance of KP is defined by a feasible set $X = \big\{x \in \{0,1\}^n : \sum_{j\in J} w_j x_j \leqslant W\big\}$, and a profit vector $c \in \mathbb{N}^n$, *i.e.*, a pair (X,c). It is naturally assumed that $\sum_{j\in J} w_j > W$ and $w_j \leqslant W$, for all $j \in J$; otherwise, the problem is obvious.

3.2 Complexity theory

This section gives a quick overview of complexity theory (see, for example, Garey and Johnson (1979) or Papadimitriou and Steiglitz (1982) for a complete review). The concepts developed in this theory are of a crucial importance to analyze problems and to design algorithms. For example, under some hypothesis such as **P**$\neq$ **NP**, it can help to answer questions such as: "Can the problem be solved with a polynomial time algorithm ?"

The complexity of an algorithm is defined with respect to the input size of the problem instances. For a given input size, it represents the worst-case behavior of the algorithm to solve an instance with the given input size. Only very large scale instances are considered in order to determine the limits of applicability of the algorithm. Therefore, only the *rate of growth* of the complexity of the algorithm is provided (Papadimitriou and Steiglitz 1982). It is for this reason that the complexity is usually characterized by the Big-O notation (Cormen et al. 2001). A function $f(n)$ belongs to the set $O(g(n))$ if there exists two positive constants k and n_0 such that $0 \leqslant f(n) \leqslant kg(n)$ for all $n \geqslant n_0$.

The complexity of an algorithm requires to measure the size of input instances. For example, the input of the {0,1}-Knapsack problem is an integer

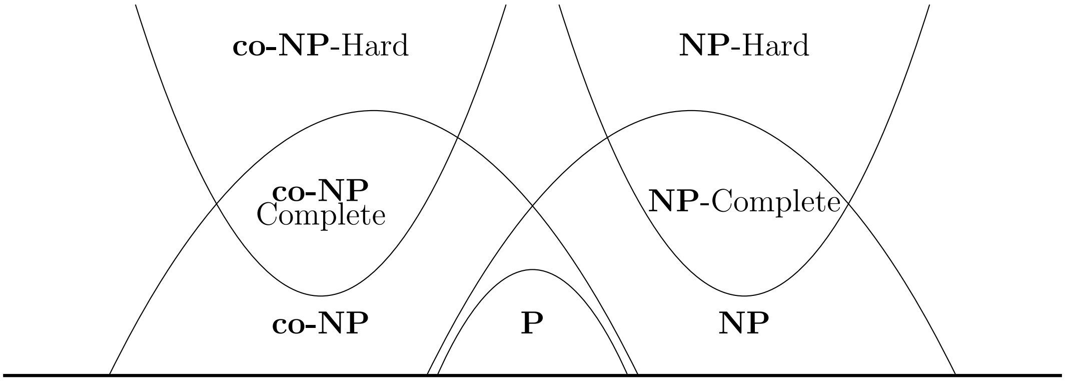

Figure 3.1: Topography of the main complexity classes.

for the number of items and a set of integers for the profits and weights. This input is encoded on a computer by a sequence of symbols over a fixed alphabet Σ such as bits. The size of the input is the length of this sequence. For example, let us encode the knapsack problem as a sequence of bits. The number of items n is encoded by $\lceil \log_2(n) \rceil$ bits. This function is in $O(\log(n))$. The weights are encoded by $\sum_{j \in J} \lceil \log_2(w_j) \rceil$ bits. This function is in $O(n \log(W))$, because it is assumed that $w_j \leqslant W$, for all $j \in J$. Similarly, the profits are encoded by a number of bits in $O(n \log(V))$, where $V = \max\{c_j : j \in J\}$. Consequently, the whole instance is encoded by a number of bits in $O(n(\log(W) + \log(V)))$.

For convenience, the theory of NP-Completeness studies *decision problems*, *i.e.*, problems that can be answered by "yes" or "no". To each optimization problem is associated a decision problem. For example, the {0,1}-Knapsack decision problem stated as follows:

The {0,1}-Knapsack Decision Problem (KDP)
INSTANCE: An instance (X, c) of KP and a $t \in \mathbb{N}$.
QUESTION: Is there an $x \in X$ with $c^\top x \geqslant t$?

If the objective function is easy to evaluate, then the decision problem is not harder than the optimization problem. Indeed, if one can solve efficiently KP, then the decision problem KDP can be solved efficiently. It consists of finding an optimal solution x^* of KP and to evaluate $c^\top x^* \geqslant t$. In this case, the optimization problem is at least as hard as the decision problem.

In this theory, problems are classified into classes such as **P**, **NP**, **co-NP**, **NP**-Complete, and **co-NP**-Complete. These classes are illustrated in Figure 3.1. Other classes may be developed, see for example Garey and Johnson (1979) for a complete review.

The class **P** is composed of all the decision problems that can be solved by a polynomial time algorithm. This class is formally defined through the use of deterministic one-tape Turing machines (DTM). Roughly speaking, a DTM program solves an instance of a decision problem by reading the input on its tape and stops on the *accept* halt-state in case of a "yes" instance, or on the *reject* halt-state in case of a “no” instance. Let L_M denote the set of strings w over an alphabet Σ such that the DTM program M accepts w, *i.e.*, the language accepted by M. A decision problem can be characterized by a language as well, *i.e.*, a set of all the input sequences that encode a “yes” instance. Hence, P is the set of languages L such that there is a polynomial time DTM program M for which $L = L_M$. A DTM program runs in polynomial time if the number of steps, for the computation to reach a halt-state, is bound by a polynomial.

The class **NP** is composed of all the decision problems for which there exists, for all “yes” instance, a concise certificate that can be checked by a polynomial time algorithm for validity. For KDP, a certificate is given by a feasible solution x such that $c^\top x \geqslant t$. This certificate is said concise, because its length is bounded by a polynomial in the size of the input instance. This class is defined formally through the use of non-deterministic Turing machines (NDTM). Roughly speaking, a NDTM program solves an instance of a decision problem by a two phase approach. During the first phase, the program guesses the certificate and write it on the tape. In the second phase, the program checks the certificate written on the tape. In case of a “yes” instance the program halts in an accepting state. Finally, **NP** is the set of languages L such that there is a polynomial time NDTM program M for which $L = L_M$. A NDTM program runs in polynomial time if the number of steps for the computation, on a string $w \in L_M$, until an accept state is reached is bound by a polynomial.

The class **co-NP** is composed of the complements of the **NP** problems. In other words, a language L belongs to **co-NP** if and only if $\bar{L}$ belongs to **NP**. Therefore, the class **co-NP** is composed of all the decision problems for which there exists, for all “no” instance, a concise certificate that can be checked by a polynomial time algorithm for validity.

A polynomial time reduction from L_1 to L_2 is denoted by $L_1 \propto L_2$. It is said that $L_1 \propto L_2$, if there exists a polynomial time DTM program M such that for all string w over Σ, the program M compute a string w' such that $w \in L_1$ if and only if $w' \in L_2$. Consequently, if L_2 belongs to **P** and $L_1 \propto L_2$, then L_1 belongs to **P** as well.

A language L is **NP**-Hard if for all $L' \in$ **NP**, $L' \propto L$. A language L is **NP**-Complete if L belongs to **NP** and **NP**-Hard classes. This implies that if there was a polynomial algorithm for an **NP**-Complete problem, then every **NP** problems could be solved in polynomial time, *i.e.*, **P** and **NP** would be the same class. For this reason, problems that are **NP**-Complete are said to be the hardest problems in **NP**.

The complexity classes **co-NP**-Hard and **co-NP**-Complete are not detailed here, because they are defined similarly.

3.3 Dynamic programming

Dynamic programming is a general approach that consists of breaking up a problem into a collection of sub-problems and to write the solution of all of them in terms of the solution of smaller sub-problems.

Let us illustrate this approach in the context of the {0,1}-knapsack problem (Kellerer et al. 1994). A dynamic programming approach for solving this problem relies on Property 1.

Property 1 (Optimal substructure)**.** *Let* $S^* \subseteq J$ *denote a feasible set of items of an instance* $\max\{c^\top x : \sum_{j\in J} w_j x_j \leqslant W\}$ *of KP and* $r \in S^*$ *an item. If* S^* *is an optimal solution, then* $S^*\backslash\{r\}$ *is an optimal solution to the knapsack sub-problem of capacity* $W - w_r$ *and item set* $J\backslash\{r\}$*.*

This property means that an optimal solution to KP contains optimal solutions to sub-problems of KP. Based on this principle, let us define the knapsack sub-problems that consider the first $i \in J$ items of J and a capacity equal to $q \in \mathbb{N}$, with $q \leqslant W$. These sub-problems consist in selecting a subset of items S in the first i items of J, such that the sum of the profits of the elements of S is maximized and the sum of weights of the same elements does not exceed q. These sub-problems are formulated by the binary integer linear programs denoted by $KP_i(q)$.

$$\begin{aligned} g_i(q) = \max \quad & \sum_{j=1}^{i} c_j x_j & \\ \text{subject to:} \quad & \sum_{j=1}^{i} w_j x_j \leqslant q & \\ & x_j \in \{0,1\}, \ j \in \{1,\dots,i\} & \end{aligned} \tag{$KP_i(q)$}$$

The function $g_i(q)$ denotes the maximum profit achievable when considering the first i items of J, with $i \in J$ and a capacity $q \in \{0,1,\dots,W\}$. Note that the original knapsack problem is to find $g_n(W)$. It is widely known that the following recursive formula solves the knapsack problem (see for example Kellerer et al. 1994).

$$g_i(q) = \begin{cases} g_{i-1}(q), & \text{if } q < w_i, \\ \\ \max\left\{g_{i-1}(q-w_i)+c_i, g_{i-1}(q)\right\}, & \text{if } q \geqslant w_i. \end{cases}$$

with $g_1(q) \leftarrow 0$ for $q = 0,1,\dots,w_1-1$ and $g_1(q) \leftarrow c_1$ for $q = w_1, w_1+1,\dots,W$.

Let us explain how this recursive formula is built. Let us assume that $g_i(q)$ is known for all $i \in \{1,2,\dots,r\}$ and capacity $q \in \{0,1,\dots,W\}$. Consider item $(r+1)$ and all sub-problems $g_{(r+1)}(q)$ with $q \in \{0,1,\dots,W\}$. Based on Property 1, either item $(r+1)$ is packed and $g_{(r+1)}(q) \leftarrow g_r(q-w_{(r+1)})+c_{(r+1)}$, or item is not packed and $g_{(r+1)}(q) \leftarrow g_r(q)$. If $w_{(r+1)}$ is greater than q, then item $(r+1)$ is too large to be packed in the knapsack $g_{(r+1)}(q)$, and therefore $g_{(r+1)}(q) \leftarrow g_r(q)$. Otherwise, the profit of packing item $(r+1)$ must be considered. If $g_r(q - w_{r+1}) + c_{r+1} > g_r(q)$, then there is a profit to place this item in the knapsack and therefore $g_{(r+1)}(q) \leftarrow g_r(q - w_{(r+1)}) + c_{(r+1)}$. Otherwise, the item is not packed and $g_{(r+1)}(q) \leftarrow g_r(q)$.

Based on this formula, Algorithm 1 computes $g_j(q)$ for all $q \in \{0, 1, \ldots, W\}$ and $j \in J$. Therefore it runs in $O(nW)$, which is a pseudo-polynomial algorithm, because W can be exponential in the input size.

Finally, the optimal solution of KP is obtained by computing the optimal solution of $KP_i(q)$. This optimal solution is denoted by $X_i(q) \subseteq J$. This set is derived form the computation of $g_i(q)$. If $g_i(q)$ is assigned to $g_{i-1}(q - w_i) + c_i$, then $X_i(q) = X_{i-1}(q - w_i) \cup \{i\}$. Otherwise, $X_i(q) = X_{i-1}(q)$.

Algorithm 1 A dynamic programming algorithm for KP.

```
for $q = 0, 1, \ldots, W$ do
    $g_0(q) \leftarrow 0$;
end for
for $j = 1, 2, \ldots, n$ do
    for $q = 0, 1, \ldots, (w_j - 1)$ do
        $g_j(q) \leftarrow g_{j-1}(q)$;
    end for
    for $q = w_j, (w_j + 1), \ldots, W$ do
        if $g_{j-1}(q - w_j) + c_j > g_{j-1}(q)$ then
            $g_j(q) \leftarrow g_{j-1}(q - w_j) + c_j$;
        else
            $g_j(q) \leftarrow g_{j-1}(q)$;
        end if
    end for
end for
```

There exists another well-known dynamic programming algorithm for solving the knapsack problem. Unlike the previous approach that performs a search in the weight space, this algorithm performs a search in the profit space. It relies on the function $h_i(v)$ that is the minimal weight of a subset of items with profit equal to v when considering the first i items. Let U denote an upper bound on the optimal solution value of the knapsack. This upper-bound is fixed by solving the linear relaxation of KP. Consequently, the following recursive formula computes the value of $h_i(v)$ for all $i \in J$ and $v \in \{0, 1, 2, \ldots, U\}$ (Kellerer et al. 1994).

$$h_i(v) = \begin{cases} h_{i-1}(v), & \text{if } v < c_i, \\ \\ \min\left\{h_{i-1}(v), h_{i-1}(v - c_i) + w_i\right\}, & \text{otherwise,} \end{cases}$$

where $h_0(0) \leftarrow 0$ and $h_0(v) \leftarrow W + 1$, for all $v \in \{1, 2, \ldots, U\}$. This recursive formula means that either item i is not packed in the knapsack and therefore $h_i(v) = h_{i-1}(v)$, or this item is packed and therefore $h_i(v) = h_{i-1}(v - c_i) + w_i$. The optimal solution of the knapsack problem is the maximal value of v such that $h_n(v) \leqslant W$. Consequently, an algorithm for solving the knapsack problem is obtained (see Algorithm 2), that runs in $O(nU)$.

Algorithm 2 A dynamic programming algorithm for KP.

```
$h_0(0) \leftarrow 0$;
for $v = 1, 2, \ldots, U$ do
    $h_0(v) \leftarrow W + 1$;
end for
for $i = 1, 2, \ldots, n$ do
    for $v = 0, 1, \ldots, (c_i - 1)$ do
        $h_i(v) \leftarrow h_{i-1}(v)$;
    end for
    for $v = c_i, (c_i + 1), \ldots, W$ do
        if $h_{i-1}(v - c_i) + w_i < h_{i-1}(v)$ then
            $h_i(v) \leftarrow h_{i-1}(v - c_i) + w_i$;
        else
            $h_i(v) \leftarrow h_{i-1}(v)$;
        end if
    end for
end for
```

3.4 Branch-and-bound algorithms

A simple approach for solving a combinatorial optimization problem could consist of enumerating all the feasible solutions. However, such an algorithm runs in $O(2^n)$. Branch-and-bound algorithms are based on a clever complete enumeration of the feasible set through the exploration of a tree (Wolsey 1998, Kellerer et al. 1994). Each node of this *branch-and-bound tree* is associated with a subset of the feasible set and the root is associated to the whole feasible set. Parent nodes and child nodes are connected as follows. Let us consider a node and its subset $X' \subseteq X$. This subset is split up into smaller ones denoted by $X_1, X_2, \ldots, X_i, \ldots, X_m \subseteq X'$, where $X_1 \cup X_2 \cup \ldots \cup X_m = X'$. These sets are the m child nodes of X' (see Figure 3.2).

This procedure is applied to each child node until the subset contains a single feasible solution. In other words, if $|X_i| = 1$, then the node is a leaf that is associated to a single feasible solution.

Each leaf node leads to a lower bound on the objective function value. An upper bound is computed on each node. Given a subset $X' \subseteq X$, an upper bound $U_{X'}$ satisfies the condition $U_{X'} \geqslant f(x)$, for all $x \in X'$.

Let z^l denote a lower bound on the objective function value. If $U_{X'} \leqslant z^l$, then there are no better solution in the sub-tree rooted at that node. Therefore, this sub-tree is pruned from the branch-and-bound tree in order to reduce the search space.

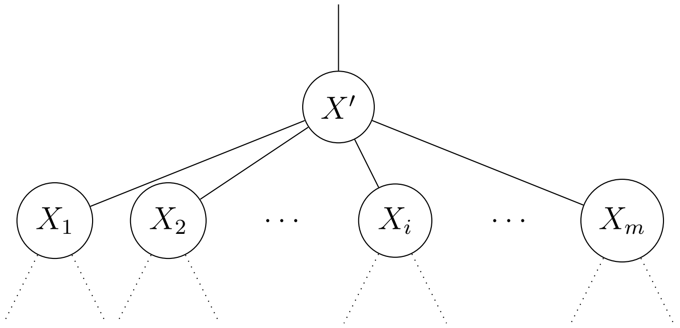

Figure 3.2: Local representation of the branch-and-bound tree with respect to a feasible set X'.

Algorithm 3 The branch-and-bound algorithm.

```
$M \leftarrow \{X\}$;
Let $z^l$ be a lower bound on $\max\{f(x) : x \in X\}$
while $M \neq \emptyset$ do
    Select a set $X'$ from $M$;
    if $|X'| = 1$ then
        Let $x$ be the single solution belonging in $X'$;
        if $f(x) > z^l$ then
            $z^l \leftarrow f(x)$;
        end if
    else
        Let $U_{X'}$ be an upper bound on $\max\{f(x) : x \in X'\}$;
        if $U_{X'} > z^l$ then
            Let split up $X'$ into $m$ subsets $X_1, X_2, \dots, X_m$;
            Add $X_1, X_2, \dots, X_m$ into $M$;
        end if
    end if
end while
```

The general branch-and-bound approach is presented in Algorithm 3. The way the subsets are added and selected from M determine how the tree is explored. There are many ways to explore this tree. The most common are the breadth-first, depth-first, and best-first manners. The first two methods are elementary tree traversal algorithms that will not be detailed in this chapter (see, for example, Cormen et al. (2001), for more details). If the set M is implemented as a stack, then the tree is explored in a depth-first manner. If the set M is implemented as a queue, then the tree is explored in a breadth-first manner. The best-first algorithm explores the tree by always considering the subset with largest upper bound first. This requires to transform only slightly Algorithm 3.

Let us illustrate the branch-and-bound method for solving the knapsack problem. In this case, $X = \{x \in \{0,1\}^n : \sum_{j \in J} w_j x_j \leqslant W\}$. An upper bound on $\max\{f(x) : x \in X\}$ is obtained by solving a linear relaxation of KP denoted by LKP.

$$\begin{array}{rll} \max & \sum_{j \in J} c_j x_j & \\ \text{subject to:} & \sum_{j \in J} w_j x_j & \leqslant W \\ & x_j & \in [0,1],\ j \in J \end{array} \tag{LKP}$$

This problem can be solved in $O(n)$ if the items are sorted with respect to their efficiency, *i.e.*, $\frac{c_j}{w_j} \geqslant \frac{c_k}{w_k}$, for all $j, k \in J$, with $j < k$ (Kellerer et al. 1994).

The branch-and-bound three is built as follows. To each node is associated a value $k \in J$ and a vector $x^0 \in \{0,1\}^k$. This pair defines a subset $X' = \big\{x \in \{0,1\}^n : \sum_{j \in J} w_j x_j \leqslant W, x_j = x_j^0, j \in \{1, 2, \ldots, k\}\big\} \subseteq X$. This subset is split up into two subsets $X_1, X_2 \subseteq X$. The first subset is defined by fixing $x_{k+1}^0 = 1$ and $k \leftarrow k+1$. The second one is defined by fixing $x_{k+1}^0 = 0$ and $k \leftarrow k+1$. These two sets are the two child nodes of X'. If a set is empty, *i.e.* $\sum_{j=1}^k x_j^0 w_j > W$, then the corresponding node is removed. Otherwise, the same procedure is applied on the set until a leaf is reached. The root node is X, with $k = 0$, and the leaves are reached when $k = n$. For each node, the linear relaxation LKP is solved in order to get an upper bound. For each leaf, the value of $f(x^0)$ is a lower bound on the optimal solution value of KP.

A complete branch-and-bound tree for the knapsack problem of Equation 3.1 is illustrated in Figure 3.3.

$$\begin{array}{rrcrl} \min & f(x) = 5x_1 & + & 7x_2 & \\ \text{subject to:} & x_1 & + & x_2 & \leqslant 1 \\ & & & x \in \{0,1\}^2 & \end{array} \tag{3.1}$$

In this example, $X_1 = \{x \in \{0,1\}^n : x_1 + x_2 \leqslant 1, x_1 = 1\}$, $X_2 = \{x \in \{0,1\}^n : x_1 + x_2 \leqslant 1, x_1 = 0\}$, $X_3 = \{(1,0)\}$, $X_4 = \{(0,1)\}$, and $X_5 = \{(0,0)\}$. The two nodes X_1 and X_2 provide two upper bounds. Their values are 5 and 7, respectively. The three leaves X_3, X_4, and X_5 provide three lower bounds. Their values are 5, 7, and 0, respectively. Therefore, the optimal solution value is 7.

3.5 Cutting plane methods

Cutting plane algorithms are very common methods for solving linear integer problems. This approach has been introduced by Dantzig et al. (1954) for solving the *traveling salesman problem* that is also well-known in combinatorial optimization. Afterward, this concept has been further extended by Gomory (1958) for solving linear integer programs.

Let us consider a linear integer problem $\max\{c^\top x : Ax \leqq b, x \in \mathbb{Z}^n\}$. The linear relaxation of this problem is $\max\{c^\top x : x \in P_0\}$, where $P_0 = \{x \in \mathbb{R}^n : Ax \leqq b\}$ is a polyhedron. Let $P_I = \mathit{conv.hull}(P_0 \cap \mathbb{Z}^n)$ denotes the *integer hull* of P, *i.e.*, the convex hull of the integer vectors in P. Hence, the linear integer

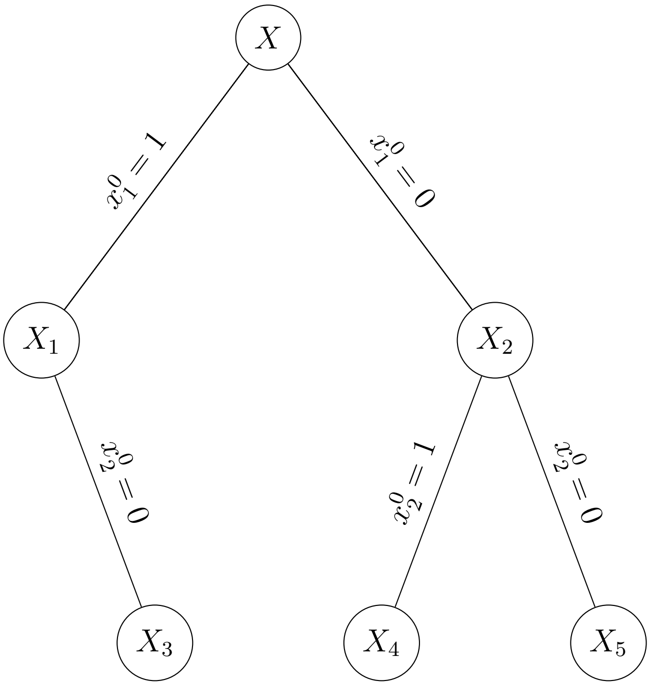

Figure 3.3: Illustration of the branch-and-bound tree for the knapsack problem of equation 3.1.

problem is equivalent to $\max\{c^\top x : x \in P_I, x \in \mathbb{Z}^n\}$. It is easy to see to see that $P_I \subseteq P_0$.

The main idea behind cutting plane methods is to find a collection of subsets $P_1, P_2, \ldots, P_i, \ldots, P_m \subseteq P_0$ such that $P_0 \supset P_1 \supset \ldots \supset P_i \supset \ldots \supset P_m \supseteq P_I$, where P_i is obtained by cutting off certain parts of $P_{(i-1)}$, for all $i \in \{1, 2, \ldots, m\}$. Each polyhedron P_i leads to a linear program $\max\{c^\top x : x \in P_i\}$ which can be solved by the simplex method. The polyhedron P_m is such that, the optimal solution of the associated linear program is an integer vector. This integer vector is obviously an optimal solution to the initial linear integer problem.

Let us illustrate this approach on the following linear integer problem (Mitchell 2009).

$$\begin{array}{rrcrl} \min & f(x) = -2x_1 & - & x_2 & \\ \text{subject to:} & x_1 & + & 2x_2 & \leqslant 7 \\ & 2x_1 & - & x_2 & \leqslant 3 \\ & & & & x \in \mathbb{Z}_+^2 \end{array} \tag{3.2}$$

The problem is illustrated in Figure 3.4 and its linear relaxation is illustrated in 3.5. The linear relaxation of Problem 3.2 is $\min\{f(x) : x \in P^0\}$, where $P_0 = \{x \in \mathbb{R}_+^2 : x_1 + 2x_2 \leqslant 7, 2x_1 - x_2 \leqslant 3\}$. The optimal solution of this relaxation is $x^* = (x_1, x_2) = (2.6, 2.2)$, and the optimal solution value is $f(x^*) = -7.4$. This relaxation must be improved in order to get an integral solution. The principle is to cut off x^* by a set of valid cutting planes, *i.e.*, to

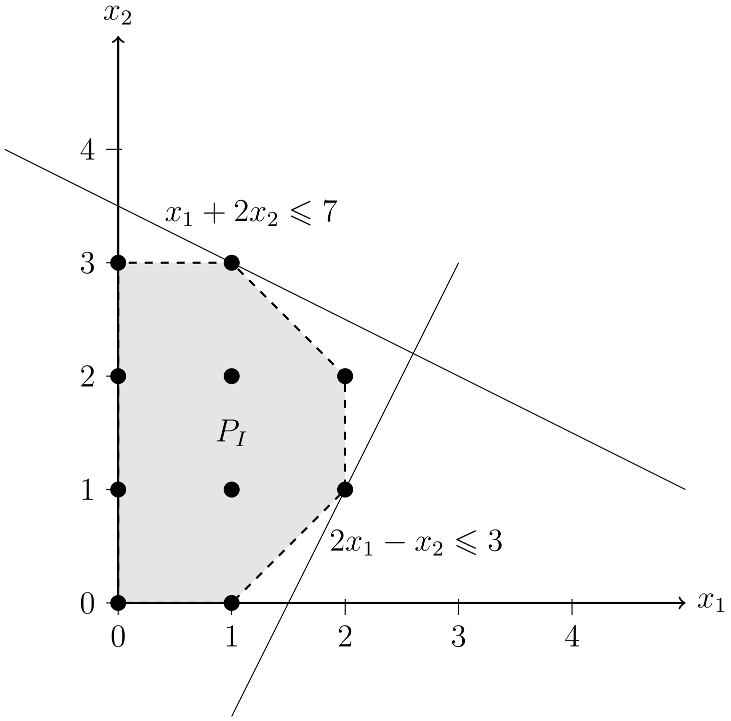

Figure 3.4: An illustration of Problem 3.2, where the integral vectors are designated by dots and the integer hull P_I is filled in gray.

add a set of constraints that are not satisfied by x^*, but satisfied by all feasible integral vectors $x \in P_I$. The inequalities $x_1 + x_2 \leqslant 4$ and $x_1 \leqslant 2$ are such valid cutting planes. By adding these two inequalities to P_0, one obtains the linear program $\min\{f(x) : x \in P^1\}$, where $P_1 = \{x \in \mathbb{R}^2_+ : x_1 + 2x_2 \leqslant 7, 2x_1 - x_2 \leqslant 3, x_1 + x_2 \leqslant 4, x_1 \leqslant 2\}$. This linear program is illustrated in Figure 3.6. The optimal solution is an integral vector $(x_1, x_2) = (2, 2)$. Therefore, this solution is an optimal solution to Problem 3.2. If the optimal solution was not integral, then the procedure would have been repeated by introducing a set of linear constraints to cut off this solution. The main steps of the cutting plane method are summarized in Algorithm 4.

Algorithm 4 A cutting plane algorithm for solving linear integer programs.

```
Let $x^*$ denotes an optimal solution to $\max\{c^\top x : x \in P\}$;
while $x^* \notin \mathbb{Z}^n$ do
    Add a set of valid cutting planes to $P$ that cut off $x^*$;
    Let $x^*$ denotes an optimal solution to $\max\{c^\top x : x \in P\}$;
end while
```

Let us note that cutting planes are often generated by using Chvátal-Gomory cuts. This method is not detailed in this section. The reader may refer to Schrijver (1986) for more detailed information on this subject.

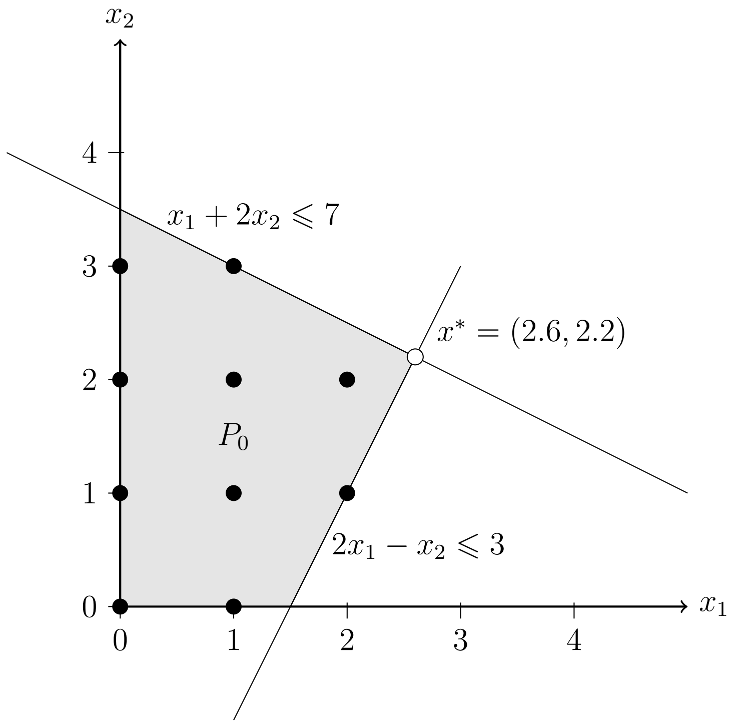

Figure 3.5: An illustration of the linear relaxation of Problem 3.2, with x^* the optimal solution.

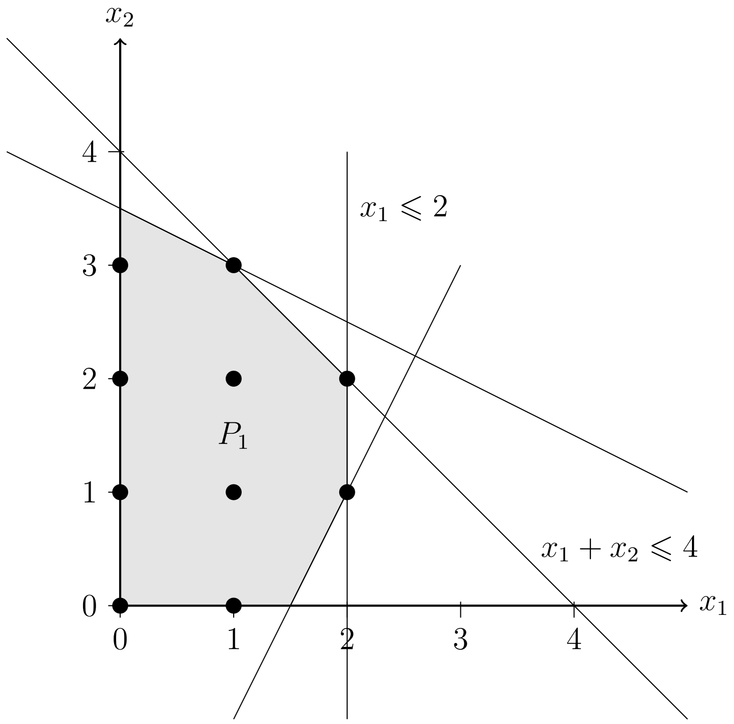

Figure 3.6: An illustration of the linear relaxation of Problem 3.2 with two additional constraints.

Chapter 4

Multi-objective combinatorial optimization

> Multi-objective optimization consists of maximizing "simultaneously" several objective functions over a set of feasible solutions. This class of problem is defined and several related problems are also considered. Various methods are presented with a special focus on the knapsack problem. We will consider only methods that are required for a good understanding of the next chapters.

4.1 Concepts, definitions and notation

Multi-objective optimization consists of maximizing "simultaneously" several objective functions over a set of feasible solutions (see, for example, Zeleny (1974), Steuer (1986), Ehrgott (2005)). The *feasible set* is denoted by $X \subseteq \mathbb{R}^n$. The outcome of each feasible solution $x \in X$ is denoted by a vector $F(x) = (f_1(x), f_2(x), \ldots, f_i(x), \ldots, f_q(x))$ composed of the outcomes of $q \geqslant 2$ objective functions $f_i : X \to \mathbb{R}$, with $i \in I$, where $I = \{1, 2, \ldots, q\}$ is the set of objective subscripts. Consequently multi-objective optimization consists of solving the problem "max"$\{F(x) : x \in X\}$, where the maximization operator is between quotation marks since there is not necessarily an outcome vector that maximize simultaneously all the objectives and that no natural total order exists on $\mathbb{R}^q$ with $q \geqslant 2$.

Without loss of generality, we assume that all the objective functions are to be maximized. Indeed, minimizing a function is equivalent to maximizing the negative of the same function.

Combinatorial problems have their counterpart in multi-objective optimization. As in single-objective optimization, a particular class of multi-objective combinatorial optimization problems (MOCO) is considered. Each instance is defined by a pair (X, C) where $X \subseteq \{x : x \in \{0,1\}^n\}$ is the feasible set and $C \in \mathbb{N}^{q \times n}$ is the so-called *profit matrix*. Each objective function

$f_i : X \to \mathbb{N}$, with $i \in I$, is defined by a row of the profit matrix with $f_i(x) = \sum_{j \in J} C_{ij} x_j = C_i x$, where C_i is the i-th row of C.

Linear optimization problems have also their counterpart in multi-objective optimization. These problems are defined as follows.

Definition 17 (Linear multi-objective optimization problem (MOLP))**.** *An instance of a multi-objective linear optimization problem is defined by* "max"$\{Cx : Ax \leqq b, x \in \mathbb{R}^n\}$*, where* $C \in \mathbb{R}^{q \times n}$*,* $A \in \mathbb{R}^{m \times n}$*, and* $b \in \mathbb{R}^m$*.*

In multi-objective optimization two spaces should be distinguished. The *decision space* is the space in which the feasible solutions are defined, and the *objective space* is the space in which the outcome vectors are defined. The image of the feasible set in the objective space is denoted by $Y = \{y \in \mathbb{R}^q : y = Cx, x \in X\}$.

Definition 18 (Ideal Solution)**.** *A feasible solution* $x \in X$ *is said to be ideal if and only if* Cx *is an ideal outcome vector.*

Definition 19 (Ideal Vector)**.** *The ideal outcome vector* $y^* \in \mathbb{R}^q$ *to an instance* (X, C) *is defined by* $y_i^* = \max\{C_i x : x \in X\}$*, for all* $i \in I$*.*

There is not necessarily an ideal solution and no natural total order exists on $\mathbb{R}^q$ with $q \geqslant 2$. Consequently, it is widely accepted to build the dominance relation on the set Y of outcome vectors. This is a binary relation that is irreflexive, asymmetric, and transitive.

Definition 20 (Dominance)**.** *Let* $y, y' \in Y$ *be two outcome vectors. It is said that* y *dominates* y' *if and only if* $y \geq y'$*.*

The dominance relation induces a partition of Y into two subsets: the set of dominated outcome vectors and the set of non-dominated outcome vectors. The set of non-dominated outcomes corresponding to an instance (X, C) is denoted by $ND(X, C)$.

Definition 21 (Non-dominated)**.** *An outcome vector* $y \in Y$ *is non-dominated if and only if there is no* $y' \in Y$ *such that* y' *dominates* y*.*

Similarly, in the decision space the concepts of efficient and non-efficient solutions can be defined. The set of efficient solutions corresponding to an instance (X, C) is denoted by $E(X, C)$.

Definition 22 (Efficiency)**.** *A solution* $x^* \in X$ *is efficient if and only if there is no* $x \in X$ *such that* $Cx \geq Cx^*$*.*

A linear integer program can be used to check whether a feasible solution x^0 is an efficient solution of (X, C).

$$\begin{aligned} \max \quad & \sum_{i\in I}\sum_{j\in J} C_{ij}x_j \\ \text{subject to:} \quad & Cx \geqq Cx^0 \\ & x \in X. \end{aligned} \tag{ND-Test}$$

Wendell and Lee (1977) have shown that, if x^* is an optimal solution to Problem ND-Test, then $x^0 \in E(X, C)$ if and only if $Cx^0 = Cx^*$.

Now, let us consider multi-objective linear combinatorial optimization problems. In such optimization problems, the non-dominated set is partitioned into two sets: the supported and the unsupported non-dominated outcome vectors sets, respectively.

Definition 23 (Supported non-dominated outcome vector)**.** *Let $y \in ND(X, C)$ denote a non-dominated outcome vector. If y is on the boundary of $Y^{\leqq} = Conv(ND(X, C) \oplus \mathbb{R}^q_{\leqq})$, then y is a supported non-dominated outcome vector. Otherwise, y is an unsupported non-dominated outcome vector.*

Some methods for computing the non-dominated outcome vectors provide only extreme non-dominated outcome vector as output. An example of such a method is presented in the next section.

Definition 24 (Supported-extreme non-dominated outcome vector)**.** *Let $y \in Y$ denote a supported non-dominated outcome vector. If y is an extreme point of $Y^{\leqq}$, then y is a supported-extreme non-dominated outcome vector. Otherwise, y is an supported non-extreme non-dominated outcome vector.*

Let us illustrate the concepts presented above on the following instance of a bi-objective optimization problem.

$$\begin{array}{rlccccccccccl} \max & f_1(x) = & & & x_2 & + & 2x_3 & + & 3x_4 & + & 4x_5 & & \\ \max & f_2(x) = & 2x_1 & + & 2x_2 & + & x_3 & & & & & & \\ \text{subject to:} & & x_1 & + & x_2 & + & x_3 & + & x_4 & + & x_5 & \leqslant & 1 \\ & & & & & & & & \multicolumn{3}{r}{x_1, x_2, \ldots, x_5} & \in & \{0,1\} \end{array} \tag{4.1}$$

The objective space of this instance is illustrated in Figure 4.1. The non-dominated set is $ND(X, C) = \{y^{(2)}, y^{(3)}, y^{(5)}\}$, where $y^{(2)}$ and $y^{(5)}$ are supported-extreme non-dominated vectors.

Let us finally point out that many MOCO problems are known to be intractable, *i.e.*, there cannot exist a polynomial time algorithm to solve them. This comes from the fact that the number of efficient solutions may be exponential in the input size. For example, even polynomial optimization problems, such as the shortest path problem, are known to be intractable in multi-objective optimization (Ehrgott 2005).

In what follows, we will present several methods for solving multi-objective combinatorial optimization problems. We only focus on methods that are required for a good understanding of inverse multi-objective optimization.

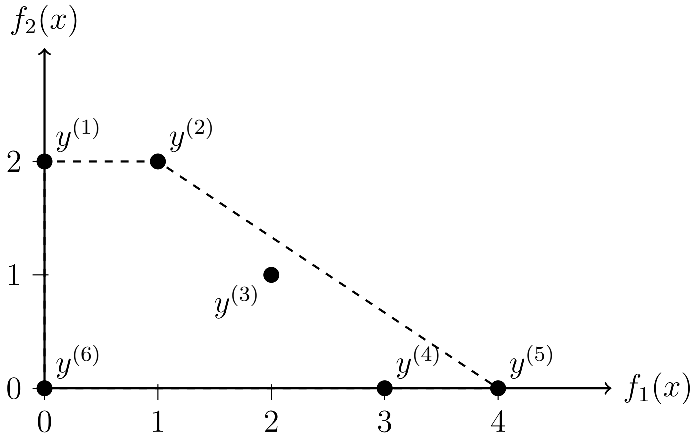

Figure 4.1: An illustration of the objective space of Problem 4.1, with the outcome of each feasible solution an the convex hull over these points.

4.2 Scalarization techniques

The set of efficient solutions of a multi-objective linear program can be obtained by solving a collection of linear programs (Geoffrion 1968). Indeed, for each efficient solution x^0 of a MOLP, there exists a vector $\lambda \in \mathbb{R}^q$, with $\lambda_i > 0$ for all $i \in I$, such that x^0 is an optimal solution of the following scalar maximum problem.

$$\begin{aligned} \max \quad & \sum_{i \in I} \lambda_i f_i(x) \\ \text{subject to:} \quad & Ax \leqq b \\ & x \in \mathbb{R}^n. \end{aligned} \tag{4.2}$$

Conversely, it is known that if $\lambda_i > 0$ for all $i \in I$, then an optimal solution of Problem 4.2 is efficient. As explained previously, in combinatiorial optimization problems there may exists unsupported non-dominated solutions. There does not exists a vector λ such that an unsupported non-dominated solutions is an optimal solution of Problem 4.2. In other words, unsupported solutions are not optimal solutions to a convex combination of the objective functions. Therefore, such a method cannot be used to find all the non-dominated solutions.

One of the most well-known technique for finding all the efficient solution is the ε-constraint method (Steuer 1986, Ehrgott 2005). This method requires to solve a collection of single-objective linear programs (Problem 4.3), where only one of the original objective function $k \in I$ is selected for maximization and the remaining $(q-1)$ functions are transformed into constraints.

$$\begin{aligned} \max \quad & f_k(x) \\ \text{subject to:} \quad & f_i(x) \geqslant \varepsilon_i, \ \text{ for all } i \in I, i \neq k \\ & x \in X. \end{aligned} \tag{4.3}$$

A feasible solution $x^0 \in X$ is efficient if and only if there exists a vector $\varepsilon \in \mathbb{R}^q$ such that x^0 is an optimal solution to Problem 4.3 for all $k \in I$.

Even though this methods looks very simple, it lacks of a well established procedure to configure Problem 4.3, *i.e.*, to fix the value of ε. Anyway, combinatorial optimization problems are usually solved by *ad hoc* methods that rely on their intrinsic structure. This is why, in the next section is presented an *ad hoc* method for solving the multi-objective knapsack problem.

4.3 Dynamic programming

The dynamic programming approaches can be applied for solving multi-objective optimization problems. Let us illustrate this method on the multi-objective {0,1}-knapsack problem. As explained in Chapter 3, the knapsack problem can be solved by performing a search in the profit space. This approach has been extended recently by Erlebach et al. (2002) to the multi-objective case.

Let UB_i denote an upper bound on the optimal solution value of the *i-th* objective function, *i.e.*, $UB_i \geqslant \max\{f_i(x) : x \in X\}$, for all $i \in I$. Let $h_k(v)$ denote the minimal weight of a subset of items with an outcome equal to $v \in \mathbb{N}^q$ when considering the first $k \in J$ items. Consequently, the following recursive formula computes the value of $h_k(v)$ for all $k \in J$ and $v \in \mathbb{N}^q$, such that $v_i \in \{0, 1, 2, \ldots, UB_i\}$, for all $i \in I$.

$$h_k(v) = \begin{cases} h_{k-1}(v), & \text{if } \exists i \in I : v_i < C_{ik}, \\ \\ \min\big\{h_{k-1}(v), h_{k-1}(v') + w_k\big\}, & \text{otherwise,} \end{cases}$$

where $v' = (v_1 - C_{1k}, v_2 - C_{2k}, \ldots, v_i - C_{ik}, \ldots, v_q - C_{qk})$. This formula is initialized as follows. If v is the null vector (*i.e.*, $v = 0$), then the value of $h_0(v)$ is fixed to 0. Otherwise, the value of $h_0(v)$ is fixed to $W + 1$ for all $v \in \{v \in \mathbb{N}^q : v_i \in \{0, 1, 2, \ldots, UB_i\}, i \in I\}$. In other words, there is no feasible solution that contains no items with an outcome equal to v. This means that the lightest knapsack without profits is the empty one. Finally, the non-dominated set requires to find all the non-dominated vectors $v \in \mathbb{N}^q$ such that $h_n(v) \leqslant W$.

Algorithm 5 computes the formula $h_k(v)$ for all $k \in J$ and $v \in \mathbb{N}^q$, such that $v_i \in \{0, 1, 2, \ldots, UB_i\}$, for all $i \in I$. Therefore, it runs in $O\big(n(\Pi_{i \in I} UB_i)\big)$, or in a reduced form in $O(n(UB_{\max})^q)$, where $UB_{\max} = \max\{UB_1, UB_2, \ldots, UB_i, \ldots, UB_q\}$. A post-processing algorithm for building the non-dominated set can be performed by any algorithm designed for problems with an explicitly given feasible set (Ehrgott 2005). For example, one could use a pairwise comparison, in which case the post-processing runs in $O\big((UB_{\max})^{2q}q\big)$, where $(UB_{\max})^q$ is the cardinality of the set on which the post-processing is performed.

Algorithm 5 A dynamic programming algorithm for the multi-objective {0,1}-knapsack problem.

```
for $v \in \mathbb{N}^q : v_i \in \{0,1,2,\dots,UB_i\}$ for all $i \in I$ do
    $h_0(v) \leftarrow W + 1$;
end for
$h_0(0) \leftarrow 0$;
for $k \in \{1,2,\dots,n\}$ do
    for $v \in \mathbb{N}^q : v_i \in \{0,1,2,\dots,UB_i\}$, for all $i \in I$ do
        if $\exists i \in I : v_i < C_{ik}$ then
            $h_k(v) \leftarrow h_{k-1}(v)$;
        else
            $v' \leftarrow (v_1 - C_{1k}, v_2 - C_{2k}, \dots, v_i - C_{ik}, \dots, v_q - C_{qk})$;
            if $h_{k-1}(v') + w_k < h_{k-1}(v)$ then
                $h_k(v) \leftarrow h_{k-1}(v') + w_k$;
            else
                $h_k(v) \leftarrow h_{k-1}(v)$;
            end if
        end if
    end for
end for
```

Chapter 5

Inverse optimization

> A quick review of inverse optimization is given in this chapter. This mathematical programming approach is motivated by two illustrative examples. Inverse optimization is then defined and several particular cases are presented. In this chapter we restrict ourselves to the problem of finding a minimal adjustment of the profit vector such that a given feasible solution becomes an optimal one. A method for solving the inverse shortest path problem is presented. This method is of a significant importance in this thesis. Finally, a previously proposed method for solving the {0,1}-knapsack problem is presented and proved to be incorrect.

5.1 Introduction

Inverse optimization takes its origins in the theory of inverse problems that arise in many fields of physics and mathematics (see, for example, Tarantola 1987). Inverse problems are loosely defined by Keller (1976) as follows: *"We call two problems inverses of each other if the formulation of each involves all or part to the solution of the other"*. One of the two problems is often called the *direct problem*, whereas the other one is called the *inverse problem*. The direct problem is usually the one that is more natural or more studied, while the inverse problem is newer and not so well understood.

Let us illustrate this concept with the direct problem stated by the following question. What are the roots of a given polynomial $p(x)$ of degree n? Let $x_1, x_2, \dots, x_r$ denote the roots of $p(x)$. An example of a related inverse problem would be stated as follows. What polynomial $p(x)$ of degree n has roots of $x_1, x_2, \dots, x_r$? Let us consider an example of such an inverse problem. Consider the polynomial $p(x) = x - 2$, where the only root is $x = 2$. The inverse problem has many solutions such as $(x - 2)$, or $k(x - 2)$ for all $k \in \mathbb{R}$, such that $k \neq 0$. As it is often the case, this inverse problem is said *ill-posed* (*i.e.*, not *well-posed*), because there exists more than one solution (Hadamard 1902).

Inverse problems have been introduced in optimization recently (see, for example, Burton and Toint 1992, Ahuja and Orlin 2001). Since then, this mathematical programming approach has been strongly developed and has shown its importance in geophysical sciences, transportation and traffic flow, among others (see, for example, Tarantola 1987, Burton and Toint 1992, Sokkalingam et al. 1999, Ahuja and Orlin 2001). Let us illustrate this concept by two applications drawn from traffic modeling and facility location.

- **Traffic Modeling** (Burton and Toint 1992)
  Shortest path techniques can be used by network planners in order to determine what are the paths taken by the users of a road network. However, these procedures do not take into account the *perceived cost* of the users. Indeed, usually users evaluate paths in terms of time, money, distance, road's quality, etc. Recovering the perceived cost is an important step in the analysis of a network users' behavior. Therefore, it is very useful to know some of the routes that are actually used (and thus considered as the optimal ones) and then incorporate this knowledge into the model, modifying the *a priori* costs in order to ensure the optimality of the paths used by the users in the modified network.

  This represents an instance of the inverse shortest path problem. Given a network, represented by a weighted graph and a given path; the question is to modify the weights, as little as possible, such that the given path becomes an optimal one.

- **Facility Location** (Heuberger 2004)
  Consider a road network and a set of clients. The facility location problem consists of installing facilities in the network in such a way that the distance to the clients is minimum. However, it can happen that the facility already exists and cannot be relocated with reasonable costs. In such a situation, one may want to modify the network as little as possible (for instance, by improving roads) in order to make optimal the current location of the facilities.

  This represents an instance of the inverse facility location problem. Given a network represented by a weighted graph, the location of a set of clients, and the location of a set of facilities; the question is to modify the weights, as little as possible, such that the given location becomes an optimal one.

There are many inverse optimization problems, however the one that is the most studied is described as follows: *"Given an optimization problem and a feasible solution to it, the corresponding inverse optimization problem consists of finding a minimal adjustment of the profit vector such that the given solution becomes optimum"* (Heuberger 2004). In the literature, this adjustment of the profit vector is generally performed under the L_1, the L_2, or the L_∞ norm.

As already stressed by (Heuberger 2004), this definition is very precise compared with what the inverse problems are. In order to cover all inverse problems, we broadly define inverse optimization as the problem of finding a minimal adjustment of an optimization instance inducing a change in the optimal solution's characteristics. For example, an inverse knapsack problem could consist of finding a minimal adjustment of weights and profits so that a given feasible solution becomes optimal. Another inverse knapsack problem could consist of finding a minimal adjustment of the capacity so that the optimal solution value is equal to a given constant. These two examples highlight the wide range of inverse problems that can be investigated. However, in this chapter, we will restrict ourselves to the more restrictive definition provided in the previous paragraph.

5.2 Inverse problems

In the context of this thesis, inverse combinatorial optimization problems are defined as follows. Let Π denote an optimization problem and $(X, f) \in \Pi$ an instance of Π. The associated inverse problem consists of determining an instance $(\mathcal{X}^*, g^*) \in \Pi$, that minimizes a cost function $\gamma : \Pi \times \Pi \to \mathbb{R}$, with respect to (X, f), such that the optimal solution of $(\mathcal{X}^*, g^*)$ satisfies a set of conditions.

The following statement is an example of one condition on the optimal solution of $(\mathcal{X}^*, g^*)$: a given feasible solution x^0 is an optimal solution of $(\mathcal{X}^*, g^*)$. In this case, the inverse problem can be modeled through the following bilevel optimization problem.

$$\begin{aligned} (\mathcal{X}^*, f^*) \in \arg\min \quad & \gamma((X, f), (\mathcal{X}, g)) \\ \text{subject to:} \quad & g(x^*) = g(x^0) \\ & x^* \in \arg\max\{g(x) : x \in \mathcal{X}\} \\ & (\mathcal{X}, g) \in \Pi \end{aligned} \tag{5.1}$$

The following statement is another example of condition: the optimal solution value of $(\mathcal{X}^*, f^*)$ is equal to a given constant $k \in \mathbb{R}$. In this case, the inverse problem can be modeled through the following bilevel optimization problem.

$$\begin{aligned} (\mathcal{X}^*, f^*) \in \arg\min \quad & \gamma((X, f), (\mathcal{X}, g)) \\ \text{subject to:} \quad & g(x^*) = k \\ & x^* \in \arg\max\{g(x) : x \in \mathcal{X}\} \\ & (\mathcal{X}, g) \in \Pi \end{aligned} \tag{5.2}$$

In this chapter we restrict ourselves to the inverse optimization problems that are defined as follows. Let Π denote a combinatorial optimization problem, $(X, c) \in \Pi$ an instance of Π, and $x^0 \in X$ a feasible solution. The associated inverse problem consists of determining a profit vector $d^* \in \mathbb{N}^n$ such that x^0 is an optimal solution of (X, d^*) and $|d^* - c|_p$ is minimum. This class of inverse problems can be modeled through the following bilevel optimization problem.

$$
\begin{aligned}
d^* \in \arg\min \quad & |d - c|_p \\
\text{subject to:} \quad & d^\top x^* = d^\top x^0 \\
& x^* \in \arg\max\{d^\top x : x \in X\} \\
& d \in \mathbb{N}^n
\end{aligned}
\tag{5.3}
$$

5.3 Polynomial optimization problems

There is a very important result that concerns every optimization problem that are polynomially solvable. Ahuja and Orlin (2001) have proved that if an optimization problem can be solved in polynomial time for every linear objective function, then inverse versions of this problem can also be solved in polynomial time if the adjustment is measured by L_1 or L_2. This result is significant, because it implies that a wide range of inverse optimization problems are polynomially solvable. However, as explained in their paper, this result cannot be used in practice for large scale instances, because it relies on the use of the ellipsoid algorithm.

Many polynomial problems have been considered in inverse optimization. In the next section the inverse shortest path will be presented. This inverse problem is of considerable interest for two main reasons. First, as stressed in the next section, it provides intuitions for solving inverse combinatorial optimization problems. Second, it is required for proving the incorrectness of the method proposed by Huang (2005) for solving the inverse knapsack problem.

5.4 The inverse shortest path

The shortest path problem is one of the most well-studied combinatorial problems in inverse optimization. Burton and Toint (1992) have studied this problem, where the adjustment is measured by the L_2 norm. They transformed this problem into a quadratic program. Zhang and Liu (1996) have transformed it into an inverse assignment problem which is solved by an instance of the assignment problem. More recently, Ahuja and Orlin (2001) have studied inverse linear programming, where the adjustment is measured by the L_∞ and L_1 norms. They have applied these results on several combinatorial problems such as the shortest path problem, the assignment problem, and the minimum cost flow problem.

Let us detail the method proposed by Ahuja and Orlin (2001) for solving the inverse shortest path problem, where the adjustment is measured by the L_1 norm. Unlike what is stated in Section 5.2, in their approach the profits can take any real value. This is only the case in this section.

Let us consider a directed graph (a digraph) denoted by $D = (V, A)$, where $V = \{1, 2, \ldots, n\}$ is the set of vertices and $A = \{(i, j) : i, j \in V\}$ is the set of arcs between pairs of vertices. It is well-known that the shortest path problem, between a source $s \in V$ and a target $t \in V$ in a digraph D, can be formulated by the following linear program.

$$
\begin{aligned}
\min \quad & \sum_{(i,j)\in A} C_{ij} X_{ij} \\
\text{subject to:} \quad & \sum_{\{j:(i,j)\in A\}} X_{ij} - \sum_{\{j:(j,i)\in A\}} X_{ji} = \begin{cases} 1, & \text{if } i = s, \\ 0, & \text{for all } i \notin \{s,t\}, \\ -1, & \text{if } i = t. \end{cases} \\
& 0 \leqslant X_{ij} \leqslant 1, \text{ for all } (i,j) \in A,
\end{aligned}
\tag{5.4}
$$

where $C \in \mathbb{R}^{n\times n}$ is the cost matrix that defines the cost of each arc (*i.e*, C_{ij} is the cost assigned to an arc (i,j)) and $X \in \{0,1\}^{n\times n}$ defines a path between s and t (*i.e*, either $X_{ij} = 1$ and (i,j) is in the path, or $X_{ij} = 0$ and (i,j) is not in the path).

The shortest path can be also formulated in a more general form, *i.e.*, a linear program of the form $\min\{c^\top x : Ax = b\}$ (Papadimitriou and Steiglitz 1982). Let us consider a graph G with m nodes and n arcs. Each arc is denoted by e_j, where c_j is the cost of e_j and x_j is equal to 1 if e_j belongs to the path from s to t and 0 otherwise, with $j \in J$. This directed graph is described by a *node-arc incidence matrix* A, where the rows and columns correspond to the nodes and arcs, respectively (Sierksma 2001). It is defined, for all $i \in \{1,2,\ldots,m\}$ and $j \in J$, as follows:

$$
A_{ij} = \begin{cases} 1, & \text{if arc } e_j \text{ leaves node } i, \\ -1, & \text{if arc } e_j \text{ enters node } i, \\ 0, & \text{otherwise.} \end{cases}
\tag{5.5}
$$

In order to ensure that the path defined by x starts from $s \in \{1,2,\ldots,m\}$ and ends to $t \in \{1,2,\ldots,m\}$, the vector b is defined, for all $i \in \{1,2,\ldots,m\}$, as follows:

$$
b_i = \begin{cases} 1, & \text{if } i = s, \\ -1, & \text{if } i = t, \\ 0, & \text{otherwise.} \end{cases}
\tag{5.6}
$$

Consequently, the shortest path problem is formulated as follows:

$$
\begin{aligned}
\min \quad & c^\top x \\
\text{subject to:} \quad & Ax = b \\
& x_j \geqslant 0, \text{ for all } j \in J.
\end{aligned}
\tag{5.7}
$$

The dual of Problem 5.7 is the following linear program.

$$
\begin{aligned}
\max \quad & \sum_{i=1}^{m} \pi_i b_i \\
\text{subject to:} \quad & \sum_{i=1}^{m} A_{ij}\pi_i + \lambda_j = c_j, \text{ for all } j \in J \\
& \pi \in \mathbb{R}^m \\
& \lambda_j \geqslant 0, \text{ for all } j \in J.
\end{aligned}
\tag{5.8}
$$

Due to the particular structure of A, this linear program can be reformulated as follows:

$$\begin{array}{rl} \max & \pi_s - \pi_t \\ \text{subject to:} & \pi_k - \pi_l + \lambda_j = c_j, \text{ for all } k, l \in \{1, 2, \ldots, m\} \text{ such that } e_j = (k, l) \\ & \pi \in \mathbb{R}^m \\ & \lambda_j \geqslant 0, \text{ for all } j \in J. \end{array} \tag{5.9}$$

With this formulation, it is easier to see that the value of π_i is equal to the length of the shortest path from node i to node t, for all $i \in \{1, 2, \ldots, m\}$.

Let us consider the complementary slackness conditions for the primal-dual pair composed of Problem 5.7 (the primal) and Problem 5.9 (the dual). Let x and (π, λ) be the primal and dual feasible solutions, respectively. Then, x and (π, λ) are both optimal if and only if for all $j \in J$ either $x_j = 0$, or $\lambda_j = 0$.

Let us consider two sets $J^0 = \{j \in J : x_j^0 = 0\}$ and $J^1 = \{j \in J : x_j^0 = 1\}$, and $F = \{j \in J : 0 < x_j^0 < 1\}$, where x^0 is a given feasible solution to Problem 5.7. Based on these sets, the complementary slackness condition implies that $\lambda_j = 0$, for all $j \in J^1$.

The inverse of Problem 5.7 consists of finding a cost vector $d \in \mathbb{R}^n$ so that x^0 becomes an optimal solution. By considering the previous condition in Problem 5.9, one obtains the inverse problem formulated by the following program.

$$\begin{array}{rl} \min & \sum_{j \in J} |d_j - c_j| \\ \text{subject to:} & \sum_{i=1}^{m} a_{ij}\pi_i = d_j, \text{ for all } j \in J^1 \\ & \sum_{i=1}^{m} a_{ij}\pi_i + \lambda_j = d_j, \text{ for all } j \in J^0 \\ & \pi \in \mathbb{R}^m \\ & \lambda_j \geqslant 0, \text{ for all } j \in J^0. \end{array} \tag{5.10}$$

The distance function between vectors c and d has to be *linearized* in order to transform Problem 5.10 into a linear program. Thus, the L_1 norm is replaced by the objective function $\sum_{j \in J}(\alpha_j + \beta_j)$ subject to $d_j - c_j = \alpha_j - \beta_j$, $\alpha_j \geqslant 0$, and $\beta_j \geqslant 0$. Consequently, Problem 5.10 can be reformulated as the following linear program.

$$
\begin{aligned}
\min \quad & \sum_{j\in J}\alpha_j+\sum_{j\in J}\beta_j \\
\text{subject to:} \quad & \sum_{i=1}^{m} A_{ij}\pi_i-\alpha_j+\beta_j=c_j,\ \text{for all } j\in J^1 \\
& \sum_{i=1}^{m} A_{ij}\pi_i+\lambda_j-\alpha_j+\beta_j=c_j,\ \text{for all } j\in J^0 \\
& \pi\in\mathbb{R}^m \\
& \lambda_j\geqslant 0,\ \text{for all } j\in J^0
\end{aligned}
\tag{5.11}
$$

The constraints in Problem 5.11 can be restated as follows:

$$
\left\{
\begin{array}{l}
-\alpha_j+\beta_j=c_j^{\pi},\ \text{for all } j\in J^1,\\
-\alpha_j+\beta_j=c_j^{\pi}-\lambda_j,\ \text{for all } j\in J^0,
\end{array}
\right.
\tag{5.12}
$$

where $c_j^{\pi}=c_j-\sum_{i=1}^{m}a_{ij}\pi_i$, for all $j\in J$. Due to the particular structure of this problem $c_j^{\pi}=c_j-\pi_k+\pi_l$, where $e_j=(k,l)$. Based on the observation we made on the value of π_i from Problem 5.9, the value of c_j^{π} can be interpreted as the detour caused by taking e_j to reach t. For this reason, it is well-known that c_j^{π} satisfies the following conditions for all $j\in J$:

$$
\left\{
\begin{array}{l}
c_j^{\pi}=0,\ \text{if } x_j^*=1,\\
c_j^{\pi}\geqslant 0,\ \text{otherwise},
\end{array}
\right.
\tag{5.13}
$$

where x^* is an optimal solution to Problem 5.7, *i.e.*, a shortest path.

Let us consider each case. If $c_j^{\pi}=0$ and $j\in J^0$, then $-\alpha_j+\beta_j=-\lambda_j$. Therefore $-\alpha_j+\beta_j=0$ and $d_j=c_j$, because $\alpha_j+\beta_j$ is minimized. If $c_j^{\pi}=0$ and $j\in J^1$, then $-\alpha_j+\beta_j=0$. Therefore. $d_j=c_j$. If $c_j^{\pi}\geqslant 0$ and $j\in J^0$, then $-\alpha_j+\beta_j=c_j^{\pi}-\lambda_j$. Therefore $-\alpha_j+\beta_j=0$ and $d_j=c_j$, because $\alpha_j+\beta_j$ is minimized. If $c_j^{\pi}\geqslant 0$ and $j\in J^1$, then $-\alpha_j+\beta_j=c_j^{\pi}$. Therefore, $\beta_j=c_j^{\pi}$ and $c_j=d_j-c_j^{\pi}$.

Based on these results, the optimal solution d^* of Problem 5.10 is obtained as follows:

$$
d_j^*=\left\{
\begin{array}{l}
c_j=d_j-c_j^{\pi},\ \text{if } x_j^0=1 \text{ and } x_j^*=0,\\
c_j=d_j,\ \text{otherwise},
\end{array}
\right.
\tag{5.14}
$$

Intuitively, this means that each time an edge e_j causes a detour, the corresponding cost is reduced in order to remove this detour. This principle is applied in the next chapter for solving the inverse knapsack problem.

Therefore, the inverse shortest path can be reduced to compute the shortest path problem which can be solved in $O(n+m\log m)$ by using the Dijkstra's algorithm when all arc costs are non-negative. However, this procedure does not ensure that the profits remain positive.

5.5 The inverse knapsack problem

The inverse $\{0,1\}$-knapsack consists of finding a minimal adjustment of the profit vector such that a given feasible set of items becomes an optimal solution. The $\{0,1\}$-knapsack problem is very interesting from a theoretical point of view. It is the simplest non-trivial integer linear program with binary variables (Pisinger 1995). This simple structure can lead to find properties that make the problem easier to solve. This is also a very general problem, because any integer program can be solved as a $\{0,1\}$-knapsack problem (Pisinger 1995, Mathews 1897). Moreover, it can appear as a subproblem of more general combinatorial optimization problems. These reasons and our knowledge of this problem have led us to consider this inverse problem as a good start before extending inverse optimization to multi-objective combinatorial optimization.

Several approaches have been suggested for dealing with such problems, where the adjustment is measured by the L_1 norm. For example, the inverse integer linear programming problem by Huang (2005), Schaefer (2009), Wang (2009), the inverse {0,1}-knapsack problem by Huang (2005), and combinatorial optimization problems by Yang and Zhang (1999). However, the said approaches are either computationally expensive, or fail in the search of inverse solutions. The methods used by Wang (2009) and Yang and Zhang (1999) are based on the cutting plane algorithm and the column generation method. However, as explained in their respective papers, those methods only guarantee to converge in a finite number of steps.

When considering the L_∞ norm one could also apply cutting plane algorithms or a column generation method but with the same drawback. A procedure based on the Newton's method has been considered by Zhang and Liu (2002), however it requires to solve a particular problem belonging to the **NP**-Hard complexity class.

The method proposed by Huang (2005) for the inverse knapsack problem is based on the reduction from the knapsack problem to the shortest path problem. However, as it will be explained in what follows, this approach is incorrect. For this purpose, let us detail the reduction from the knapsack to the shortest path.

A reduction from the {0,1}-knapsack problem to the shortest path problem

It is well-known that the {0,1}-knapsack problem can be formulated as a shortest path on a directed acyclic graph (Schrijver 1986). In order to simplify this approach, we assume that $W > 0$, and $c_j, w_j > 0$ holds for all $j \in J$. This formulation is based on the dynamic programming approach presented in Chapter 3. Let us recall that it considers a function $g_j(q)$ that is the maximum profit achievable when considering the first j items and a capacity equal to q. The recursive formula, that computes this function, can be implemented by a directed graph $D = (V, A)$. In this digraph, the set of all the vertices V is defined as follows:

$$V = (0, 1, \ldots, n) \times (0, 1, \ldots, W-1, W), \tag{5.15}$$

where $\{0, 1, \ldots, W-1, W\}$ contains all the possible value of feasible solution's weight. A vertex $(j-1, q)$ represents the solution when considering the $j-1$ first items and a capacity of q. As detailed in dynamic programming, from this state, there are two possibilities, *i.e.*, either item j is packed in the solution, or item j remains outside of the knapsack. These choices can be represented by the set of arcs denoted by A.

$$A = \Big\{((j-1, q), (j, q')) : (q' - q \in \{0, w_j\}) \wedge j \in \{1, 2, \ldots, n\}\Big\}, \tag{5.16}$$

where an arc $((j-1, q), (j, q+w_j))$ represents the decision of including item j in the knapsack and an arc $((j-1, q), (j, q))$ represents the decision of not including it. This graph is usually represented by using $(n+1)$ layers, where each vertex (j, q) is in the *j-th* layer, for all $j = 0, \ldots, n$ and $q \in \{0, 1, \ldots, W\}$ (See Figure 5.1). In this configuration, each arc connects two vertices from two adjacent layers, because if there is an arc between (j, q) and (k, q'), then $j = (k-1)$.

Finally, the profit of including an item j to the knapsack is provided by the arc length $l : A \rightarrow \mathbb{N}$.

$$l((j-1, q), (j, q')) = \left\{ \begin{array}{ll} c_j, & \text{if } q' - q = w_j, \\ 0, & \text{otherwise.} \end{array} \right. \tag{5.17}$$

Each arc is the link between a solution of weight q using the $(j-1)$ first items to a solution of weight q' using the j first items. These pairs of solutions can be linked if their respective weights allows this consistently with the dynamic programming recursive formula. Therefore, it is easy to see that any directed path P from $(0, 0)$ to (n, q) for some $q \leqslant W$ yields to a feasible solution x defined as follows.

$$x_j = \left\{ \begin{array}{ll} 0, & \text{if } \exists q \in \mathbb{N} : ((j-1, q), (j, q)) \in P, \\ 1, & \text{if } \exists q \in \mathbb{N} : ((j-1, q), (j, q+w_j)) \in P. \end{array} \right. \tag{5.18}$$

Consequently, the optimal solution is the longest path between $(0, 0)$ and (n, q), with $q \leqslant W$. In order to find the optimal solution, a second graph denoted by $D' = (V', A')$ is built as follows.

$$\left\{ \begin{array}{l} V' = V \cup \{t\}, \\ A' = A \cup \big\{((n, q), t) : q \in \{0, 1, \ldots, W-1, W\}\big\}, \\ l((n, q), t)) = 0 \text{ for all } q \in \{0, \ldots, W-1, W\}. \end{array} \right. \tag{5.19}$$

It is easy to see that finding the optimal solution is equivalent to finding the longest path from the source $(0, 0)$ to the target vertex t in $D' = (V', A')$. As there is no cycle in this graph, the longest path can be found by using traditional shortest path algorithms with a length function l' such that $l'(a) = -l(a)$, for all $a \in A'$

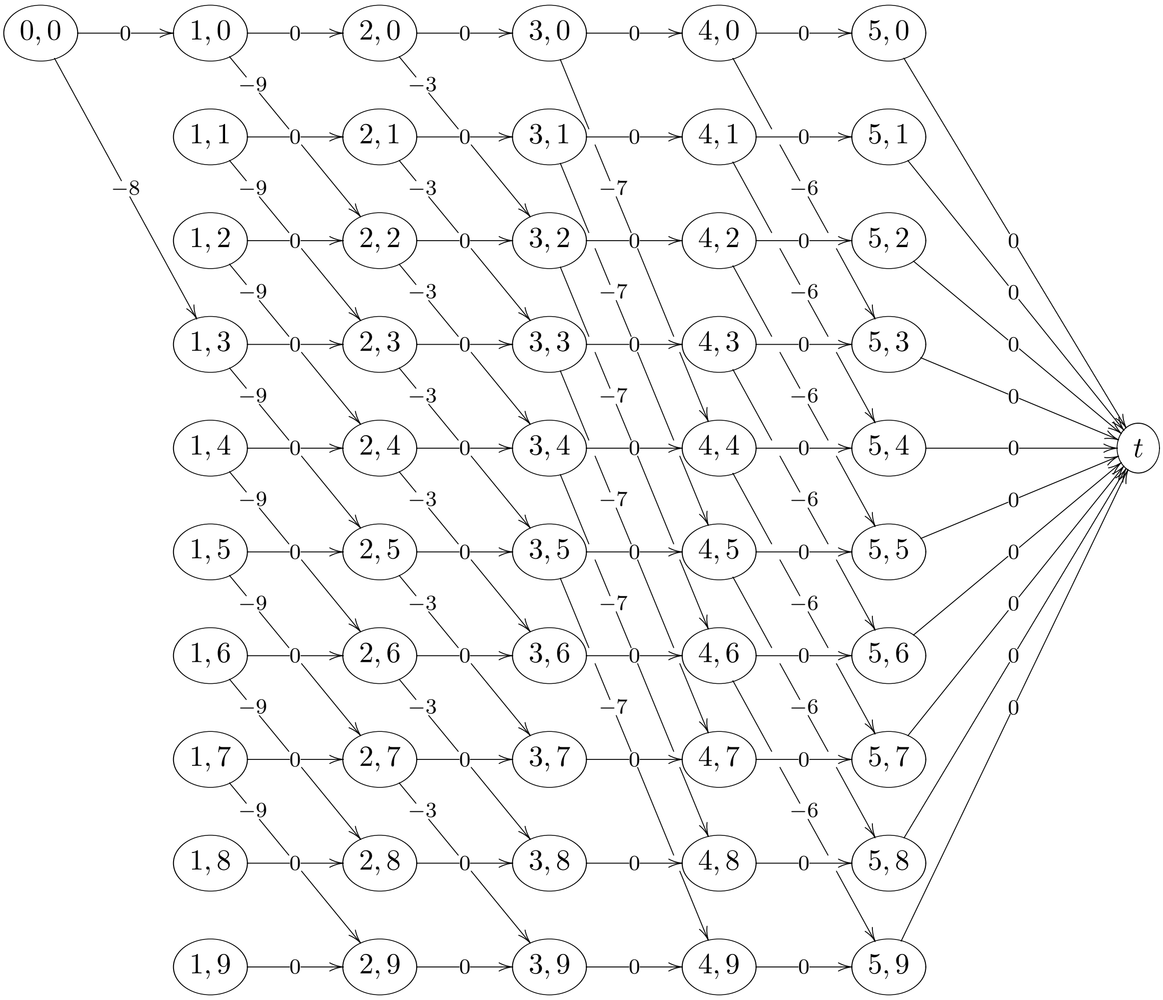

Figure 5.1: Network model of the knapsack problem from equation 5.20

Let us consider the following knapsack problem as an illustrative example:

$$\begin{array}{rll} \max & f(x) = & 8x_1 + 9x_2 + 3x_3 + 7x_4 + 6x_5 \\ \text{subject to:} & & 3x_1 + 2x_2 + 2x_3 + 4x_4 + 3x_5 \leqslant 9 \\ & & x_i \in \{0,1\},\ i = 1,\ldots,5, \end{array} \tag{5.20}$$

where the cost vector c is $(8, 9, 3, 7, 6)$. This leads to build the network model presented in Figure 5.1.

An incorrect approach for solving the inverse {0,1}-knapsack problem

Based on this reduction, Huang (2005) concluded that the inverse {0,1}-knapsack can be reduced to solve the inverse shortest path on the knapsack's network model. However, this conclusion is not valid, because the network associated with the knapsack problem is a very particular one and therefore any modification on the weights should be applied with caution. For example, in Figure 5.1 if the length of $((2,0),(3,2))$ is modified, this actually means that the third item's profit is modified. Therefore, the length of many arcs such as $((2,1),(3,3))$ and $((2,2),(3,4))$ must also be modified in such a way that the network remains correct. In other words, the inverse {0,1}-knapsack problem can be solved as the inverse shortest path problem, but only with additional constraints, which was not the case in the work by Huang (2005).

Due to the lack of an *ad hoc* approach for solving the inverse knapsack problem, the next chapter is dedicated to the study of this problem. Several methods are proposed and a complexity analysis is presented.

Chapter 6

The inverse {0,1}-knapsack problem

The inverse {0,1}-knapsack problem consists of finding a minimal adjustment of the profit vector such that a given feasible set of items becomes an optimal solution. In this chapter, two models are considered. In the first, the adjustment is measured by the Chebyshev norm. A pseudo-polynomial time algorithm is proposed to solve it. In the second, the adjustment is based on the Manhattan norm. This model is reduced to solve a linear integer program. While the first problem is proved to be **co-NP**-Complete, the second is **co-NP**-Hard and strong arguments are against its **co-NP**-Completeness. For both models, a bilevel linear integer programming formulation is also presented. Numerical results from computational experiments to assessing the feasibility of these approaches are reported.

6.1 Introduction

In this chapter, the inverse $\{0, 1\}$-knapsack problem is studied. The adjustment is first measured under the L_∞ norm. The problem is theoretically studied and a proof of its **co-NP**-Completeness is provided. Combinatorial arguments are used to present a pseudo-polynomial time algorithm. The inverse $\{0, 1\}$-knapsack problem under the L_1 norm is then considered. The problem is theoretically studied and a proof of its **co-NP**-Hardness is also provided. Next, as elegantly proposed by Ahuja and Orlin (2001), optimality conditions are used to formulate this problem as an integer linear program. Such conditions are described in the dynamic programming algorithm for the $\{0, 1\}$-knapsack problem. This implies that, if the profit vector is integer-valued, the inverse problem is reduced to solve an integer linear program. Otherwise, the inverse problem is solved by the linear relaxation of the integer linear program, which can be solved with a pseudo-polynomial time algorithm. Unlike the algorithm proposed by Huang (2005), our method can be used to solve all problem instances. Furthermore, comparing to the approach proposed by Schaefer (2009), this formulation can significantly reduce the number of variables.

This chapter also proposes a bilevel integer linear programming formulation for solving the inverse $\{0,1\}$-knapsack problem. This implies that branch-and-bound (Moore and Bard 1990) and branch-and-cut (DeNegre and Ralphs 2009) algorithms can be used to solve it.

The methods proposed in this chapter have been implemented and computational experiments were made on a large set of randomly generated instances. The purpose of the experiments is to determine how the performance of the algorithms changes with different inputs in order to assess their practical applicability.

This chapter is organized as follows: in Section 6.2, a pseudo-polynomial time algorithm is presented to solve the inverse $\{0,1\}$-knapsack problem under the L_∞ distance. In Section 6.3, an integer linear program for solving the inverse problem is stated in the context of the L_1 distance. In Section 6.4, a bilevel programming approach is developed. In Section 6.5, computational experiments with knapsack problems under the L_∞ and the L_1 norms are presented.

6.2 The Inverse $\{0,1\}$-Knapsack Problem under L_∞

This section deals with the problem and some theoretical results, which lead to proposing an algorithm for solving the inverse $\{0,1\}$-knapsack problem under the L_∞ distance.

Problem Definition

Let (X, c) denote an instance of the $\{0,1\}$-knapsack problem and $x^0 \in X$ a feasible solution. The L_∞ *inverse* $\{0,1\}$*-knapsack problem* (IKP-∞) can be stated as follows:

$$\begin{array}{rl} d^* \in \arg\min & \max_{j\in J}\{|c_j - d_j|\} \\ \text{subject to:} & d^\top x^* = d^\top x^0 \\ & x^* \in \arg\max\{d^\top x : x \in X\} \\ & d \in \mathbb{N}^n \end{array} \tag{IKP-∞}$$

IKP-∞ is a bilevel optimization problem which determines a profit vector $d^* \in \mathbb{N}^n$, which minimizes the L_∞ distance with respect to c and such that x^0 is an optimal solution of the modified knapsack problem (X, d^*).

Theoretical Results

We start by analyzing the nature of some optimal solutions of IKP-∞. Based on a partition of J defined by $J^0 = \{j \in J : x^0_j = 0\}$ and $J^1 = \{j \in J : x^0_j =$

$1\}$, the first theorem establishes that an optimal solution d^* can be built by increasing c_j, for all $j \in J^1$ and by decreasing c_j, for all $j \in J^0$.

Theorem 1. *There exists an optimal solution* $d^* \in \mathbb{N}^n$ *of IKP-*∞ *where* $\forall j \in J^1 : d^*_j \geqslant c_j$ *and* $\forall j \in J^0 : d^*_j \leqslant c_j$.

Proof. Let $d \in \mathbb{N}^n$ denote any optimal solution of IKP-∞, $J^{0>} = \{j \in J^0 : d_j > c_j\}$, $J^{1<} = \{j \in J^1 : d_j < c_j\}$. Consider a solution d^* of IKP-∞ defined as follows, for all $j \in J$:

$$d^*_j = \begin{cases} c_j, & \text{if } j \in \{J^{1<} \cup J^{0>}\}, \\ \\ d_j, & \text{otherwise.} \end{cases} \tag{6.1}$$

The constraints set of IKP-∞ implies that $d^\top x^0 \geqslant d^\top x$ for all feasible solutions $x \in X$. Let us show that for all $x \in X$, if $d^\top x^0 \geqslant d^\top x$, then $d^{*\top} x^0 \geqslant d^{*\top} x$. This is equivalent to show that the following condition holds: if $d^\top(x^0 - x) \geqslant 0$, then $d^{*\top}(x^0 - x) \geqslant 0$. The profit vector d^* is introduced in the first inequality as follows, $(d - d^* + d^*)^\top (x^0 - x) \geqslant 0$, which leads to write: $d^{*\top}(x^0 - x) \geqslant (d^* - d)^\top (x^0 - x)$. From Equation 6.1 and the definition of J^1, one may deduce that, for all $j \in J^1$, $(x^0 - x)_j \geqslant 0$ and $(d^*_j - d_j) \geqslant 0$. Similarly, for all $j \in J^0$, $(x^0 - x)_j \leqslant 0$ and $(d^*_j - d_j) \leqslant 0$. Therefore, $(d^* - d)^\top (x^0 - x) \geqslant 0$, and consequently $d^{*\top}(x^0 - x) \geqslant 0$. Therefore, d^* is an optimal solution of IKP-∞, and the theorem is proved.

□

Let us define a vector $d^k \in \mathbb{N}^n$ of distance at most k from vector c with respect to the L_∞ norm.

Definition 25 (Vector d^k). *Given* $k \in \mathbb{N}$, *define for all* $j \in J$,

$$d^k_j = \begin{cases} \max\{0, c_j - k\}, & \text{if } x^0_j = 0, \\ \\ c_j + k, & \text{if } x^0_j = 1. \end{cases}$$

The following theorem provides an optimality condition for d^k based on the value of k.

Theorem 2. *If* d^* *denotes an optimal solution of IKP-*∞, *with* $k = \max_{j \in J}\{|c_j - d^*_j|\}$, *then* d^k *is also an optimal solution of IKP-*∞.

Proof. Let d^* denote an optimal solution of IKP-∞ such that $\max_{j \in J}\{|c_j - d^*_j|\} = k$, and $J^{<k} = \{j \in J : (|c_j - d^*_j| < k) \wedge (d^*_j \neq 0)\}$. Based on Theorem 1, it can be assumed that $\forall j \in J^1 : d^*_j \geqslant c_j$ and $\forall j \in J^0 : d^*_j \leqslant c_j$. Therefore, if $|J^{<k}| = 0$, then $d^* = d^k$. Let us assume that $|J^{<k}| \geqslant 1$. Let $J^{1,<k} = J^1 \cap J^{<k}$ and $J^{0,<k} = J^0 \cap J^{<k}$. It is easy to see that the value of d^*_j can be increased for all $j \in J^{1,<k}$ and decreased for all $j \in J^{0,<k}$ without altering the optimality

of x^0. Let us detail this result. By hypothesis, $d^{*\top} x^0 \geqslant d^{*\top} x$, for all $x \in X$. This can be written as follows:

$$\sum_{j\in J\setminus J^{<k}} d^*_j x^0_j + \sum_{j\in J^{1,<k}} d^*_j x^0_j \geqslant \sum_{j\in J\setminus J^{<k}} d^*_j x_j + \sum_{j\in J^{1,<k}} d^*_j x_j + \sum_{j\in J^{0,<k}} d^*_j x_j$$

Based on the definition of d^k,

$$\sum_{j\in J\setminus J^{<k}} d^k_j x^0_j + \sum_{j\in J^{1,<k}} d^k_j x^0_j \geqslant \sum_{j\in J\setminus J^{<k}} d^k_j x_j + \sum_{j\in J^{1,<k}} d^k_j x_j + \sum_{j\in J^{0,<k}} d^*_j x_j.$$

Since $\sum_{j\in J^{0,<k}} d^*_j x_j \geqslant \sum_{j\in J^{0,<k}} d^k_j x_j$, one obtains:

$$\sum_{j\in J\setminus J^{<k}} d^k_j x^0_j + \sum_{j\in J^{1,<k}} d^k_j x^0_j \geqslant \sum_{j\in J\setminus J^{<k}} d^k_j x_j + \sum_{j\in J^{1,<k}} d^k_j x_j + \sum_{j\in J^{0,<k}} d^k_j x_j$$

Therefore, for all $x \in X$:

$$\sum_{j\in J} d^k_j x^0_j \geqslant \sum_{j\in J} d^k_j x_j$$

Consequently d^k is an optimal solution of IKP-∞ and the theorem is proved. □

As a corollary of the theorem, an optimal solution of IKP-∞ can be built on the basis of the distance between vectors c and d^*. Therefore, IKP-∞ can be reduced to finding the L_∞ distance between d^* and c, which is easy to compute since it is given by the minimal value of k where x^0 is an optimal solution with respect to d^k. In order to reduce the search domain, an upper bound on the distance is provided in the following lemma.

Lemma 1. *Let $D_\infty \in \mathbb{N}$ denote the optimal solution value of IKP-∞. Then, $D_\infty \leqslant C = \max_{j\in J^0}\{c_j\}$*

Proof. It is always possible to build a vector $d \in \mathbb{N}^n$ with $\max_{j\in J}\{|c_j - d_j|\} = \max_{j\in J^0}\{c_j\}$ such that $d^\top x^0 = \max\{d^\top x : x \in X\}$. The vector is defined as follows, $\forall j \in J^1 : d_j = c_j$ and $\forall j \in J^0 : d_j = 0$. It is easy to see that for all $x \in X$, one obtains $d^\top x^0 \geqslant d^\top x$ and $\max_{j\in J}\{|c_j - d_j|\} = \max_{j\in J^0}\{c_j\}$. This concludes the proof. □

Naturally, the value of k can be increased without altering the optimality of x^0 with respect to vector d^k. This is expressed in Theorem 3.

Theorem 3. *If d^k satisfies $d^{k\top} x^0 = \max\{d^{k\top} x : x \in X\}$, then for all $k' \in \mathbb{N}$, with $k' > k$, one obtains $d^{k'\top} x^0 = \max\{d^{k'\top} x : x \in X\}$.*

Proof. Similarly to the proof of Theorem 2, it is easy to see that the value of d^k_j can be increased for all $j \in J^1$ and decreased for all $j \in J^0$ without altering the optimality of x^0. Consequently $d^{k'}$ is a feasible solution of IKP-∞ and the theorem is proved. □

Complexity Results

As with many combinatorial optimization problems, an important issue is to analyze the complexity of the IKP-∞ problem. We shall start by defining the corresponding decision problem.

The Inverse $\{0,1\}$-Knapsack Decision Problem under L_∞ (IKDP-∞)
INSTANCE: An instance (X, c) of KP, a feasible solution $x^0 \in X$ and a $k \in \mathbb{N}^0$.
QUESTION: Is there a vector $d \in \mathbb{N}^n$ such that $\max_{j \in J}\{|c_j - d_j|\} \leqslant k$, $d^\top x^* = d^\top x^0$ and $x^* \in \arg\max\{d^\top x : x \in X\}$?

IKP-∞ can be solved by using a *binary search* for the optimal solution value through a certain number of calls to IKDP-∞. *Sketch of the algorithm.* Based on Lemma 1, it is known that the optimal solution value must be between $a \leftarrow 0$ and $b \leftarrow C$. Call IKDP-∞ with $k \leftarrow \lfloor (a+b)/2 \rfloor$. If the answer is "Yes", set $b \leftarrow k$; otherwise, set $a \leftarrow k+1$, and repeat the process. When $a = b$, the optimal solution value is obtained. The number of calls to IKDP-∞ is bounded from above by $\log_2 C$, which is polynomial in input length. Furthermore, based on Theorem 2, it is also known that the optimal solution of IKP-∞ can be computed in polynomial time based on the optimal solution value of IKDP-∞. Therefore, if IKDP-∞ can be solved in polynomial time, then IKP-∞ can also be solved in polynomial time.

It is easy to verify that if IKDP-∞ can be solved in polynomial time, then IKP-∞ can also be solved in polynomial time by applying a binary search.

Theorem 4. *IKDP-∞ is **co-NP**-Complete.*

Proof. To prove the Theorem, let us introduce the decision problem $\overline{\text{IKDP-}\infty}$.

$\overline{\text{IKDP-}\infty}$
INSTANCE: An instance (X, c) of KP, a feasible solution $x^0 \in X$ and a $k \in \mathbb{N}$.
QUESTION: Is there a feasible solution $x \in X$ such that $d^{k\top} x > d^{k\top} x^0$?

Let us prove the **NP**-Completeness of this decision problem. It is easy to see that $\overline{\text{IKDP-}\infty}$ belongs to **NP**, since a nondeterministic Turing machine has only to guess a subset of J represented by an incidence vector x and check in polynomial time that $x \in X$ and $d^{k\top} x > d^{k\top} x^0$.

Consider the {0,1}-Knapsack decision problem stated as follows:

The {0,1}-Knapsack Decision Problem (KDP)
INSTANCE: An instance (X, c) of KP and a $t \in \mathbb{N}$.
QUESTION: Is there an $x \in X$ with $c^\top x \geqslant t$?

The **NP**-Hardness of $\overline{\text{IKDP-}\infty}$ is provided by a polynomial time reduction from KDP to $\overline{\text{IKDP-}\infty}$. An instance of KDP is transformed into an instance of

$\overline{\text{IKDP-}\infty}$ by adding an item to the set J, with $c_{n+1} = t-1$, $w_{n+1} = W$, $x_j^0 = 0$, for $j = 1, \ldots, n$, $x_{n+1}^0 = 1$, and $k = 0$. It is easy to see that the reduction is polynomial. The correctness of this reduction is proved in what follows. For any given "Yes" instance of KDP, there exists an $x \in X$ such that $c^\top x \geqslant t$. Therefore, in the corresponding instance of $\overline{\text{IKDP-}\infty}$, there exists an $x \in X$ such that $d^{k\top} x > d^{k\top} x^0$. For any given "No" instance of KDP, $c^\top x < t$ for all $x \in X$. Therefore, in the corresponding instance of $\overline{\text{IKDP-}\infty}$, there is no $x \in X$ such that $d^{k\top} x > d^{k\top} x^0$. This proves the reduction. Thus, $\overline{\text{IKDP-}\infty}$ is **NP**-Complete.

Next, let us prove that IKDP-∞ is the complement of $\overline{\text{IKDP-}\infty}$. Any "Yes" instance of $\overline{\text{IKDP-}\infty}$ is a "No" instance of IKDP-∞. This is deduced from Theorems 2 and 3. By the definition of d^k, any "No" instance of $\overline{\text{IKDP-}\infty}$ is a "Yes" instance of IKDP-∞. The complement of an **NP**-Complete problem is **co-NP**-Complete (see, for example, Garey and Johnson (1979)). Consequently, IKDP-∞ is a **co-NP**-Complete problem and the theorem is proved. □

A pseudo-polynomial time algorithm is provided in what follows, which implies that IKP-∞ is fortunately a weakly **co-NP**-Complete problem.

A Pseudo-polynomial Time Algorithm

A pseudo-polynomial time algorithm for computing an optimal solution of IKP-∞ is proposed in this section.Thanks to Lemma 1, it is known that the distance between vectors d^* and c is bound from above by $C = \max_{j \in J}\{(1 - x_j^0)c_j\}$. The algorithm consists of finding the minimal value $k \in \{0, 1, \ldots, C\}$, so that x^0 is an optimal solution of the knapsack problem (X, d^k).

Sketch of the algorithm. Start with $k \leftarrow 0$. Compute a profit vector d^k. If the knapsack problem with d^k provides an optimal solution x^* such that $d^{k\top} x^* = d^{k\top} x^0$, then stop. Set $d^* \leftarrow d^k$. Otherwise, repeat the previous step with $k \leftarrow k + 1$. The correctness of this algorithm relies directly from Lemma 1 and Theorem 2.

The following pseudo-polynomial time algorithm (see Algorithm 6 for the pseudo-code) can be established with this procedure. It makes use of a solver denoted by $KP(X, c)$, which gives the optimal solution value of the knapsack problem (X, c).

Algorithm (6) runs in $O(nWC)$. Let us detail how this complexity is determined. The first loop (line 3) runs at most C times. The loop has two parts: a second loop (line 4) and a call to the knapsack solver KP (line 11). The second loop runs in $O(n)$ and the call to the knapsack solver runs in $O(nW)$ by using a dynamic programming approach (see, for example, Kellerer et al. (1994)). Therefore, these two parts run in $O(nW)$ and the whole algorithm runs in $O(nWC)$. Due to the call to the knapsack solver and C (that can be exponential in input size), we have a pseudo-polynomial time algorithm.

The running time complexity of this approach can be reduced to $O(nW \log C)$ by using a *binary search*, because as stated in Theorem 3, the value of k can be increased without altering the optimality of x^0 with respect to vector d^k.

Algorithm 6 Compute an optimal solution d^* of IKP-∞.

```
$C \leftarrow \max_{j \in J}\{(1 - x_j^0)c_j\}$;
OPT $\leftarrow KP(X, c)$;
for all ($k = 1$ to $C$) and (OPT $> d^{k\top}x^0$) do
    for all ($j = 1$ to $n$) do
        if ($x_j^0 = 0$) then
            $d_j^k \leftarrow \max\{0, c_j - k\}$;
        else if ($x_j^0 = 1$) then
            $d_j^k \leftarrow c_j + k$;
        end if
    end for
    OPT $\leftarrow KP(X, d^k)$;
end for
$d^* \leftarrow d^k$;
```

Sketch of the algorithm. Based on Lemma 1, it is known that the optimal solution must be between $a \leftarrow 0$ and $b \leftarrow C$. Build a vector d as stated in Theorem 2 by using the distance $k \leftarrow a + \lfloor (b - a)/2 \rfloor$. If x^0 is an optimal solution for the resulting vector, set $b \leftarrow k$; otherwise, set $a \leftarrow k + 1$ and repeat the process. When $a = b$, the optimal solution is obtained. See Algorithm 7 for a pseudo-code of this procedure.

Algorithm 7 Compute an optimal solution d^* of IKP-∞.

```
 1: $a \leftarrow 0$;
 2: $b \leftarrow C$;
 3: while $a \neq b$ do
 4:     $k \leftarrow a + \lfloor (b - a)/2 \rfloor$;
 5:     for all ($j = 1$ to $n$) do
 6:         if ($x_j^0 = 0$) then
 7:             $d_j^k \leftarrow \max\{0, c_j - k\}$;
 8:         else if ($x_j^0 = 1$) then
 9:             $d_j^k \leftarrow c_j + k$;
10:         end if
11:     end for
12:     OPT $\leftarrow KP(X, d^k)$;
13:     if $OPT = d^{k\top}x^0$ then
14:         $b \leftarrow k$;
15:     else
16:         $a \leftarrow k + 1$;
17:     end if
18: end while
19: $d^* \leftarrow d^a$;
```

Algorithm (7) runs in $O(nW \log C)$. Let us detail how this complexity is determined. The first loop (line 3) runs at most $\log C$ times. The loop has two parts: a second loop (line 5) and a call to the knapsack solver KP (line 11). The second loop runs in $O(n)$ and the call to the knapsack solver runs in $O(nW)$ by using a dynamic programming approach (see, for example, Kellerer et al. (1994)). Therefore, these two parts run in $O(nW)$ and the whole algorithm runs in $O(nW \log C)$. Due to the call to the knapsack solver and C (that can be exponential in input size), we have a pseudo-polynomial time algorithm.

6.3 The Inverse $\{0,1\}$-Knapsack Problem under L_1

This section deals with the problem and several theoretical results, which lead to designing an integer linear programming model for solving the inverse $\{0,1\}$-knapsack problem under the L_1 distance. To our best knowledge, this is the first formulation that solves all the problem instances with a pseudo-polynomial number of variables and constraints. Indeed, the integer linear programming formulation of Schaefer (2009) suggested to solve inverse integer programming problems is defined by an exponential number of constraints.

Problem Definition

Let $x^0 \in X$ denote a feasible solution. The L_1 *inverse* $\{0,1\}$*-knapsack problem* (IKP-1) can be stated as follows:

$$\begin{array}{rl} \min & \sum_{j \in J} |c_j - d_j| \\ \text{subject to:} & d^\top x^* = d^\top x^0 \\ & x^* \in \arg\max\{d^\top x : x \in X\} \\ & d \in \mathbb{N}^n \end{array} \tag{IKP-1}$$

IKP-1 is a bilevel optimization problem that determines a profit vector $d^* \in \mathbb{N}^n$, which minimizes the L_1 distance with respect to c such that x^0 is an optimal solution of the modified knapsack problem (X, d^*).

Some Theoretical Results

We shall start by introducing a lower bound on the L_1 distance between vectors c and d^* inspired by the upper bound proposed by Ahuja and Orlin (2002) for the shortest path problem.

Lemma 2. *Let d^* denote an optimal profit vector for IKP-1, x^0 an optimal solution for (X, d^*) and $x' \in X$. Then, a lower bound on the optimal solution value of IKP-1 is given by*

$$|d^* - c|_1 = \sum_{j \in J} |d^*_j - c_j| \geqslant (c^\top x' - c^\top x^0)$$

Proof.

$$\begin{aligned}|d^* - c|_1 &\geqslant (d^{*\top} - c^\top)(x^0 - x')\\ &= d^{*\top}x^0 - d^{*\top}x' - c^\top x^0 + c^\top x'\\ &= (d^{*\top}x^0 - d^{*\top}x') + (c^\top x' - c^\top x^0)\\ &\geqslant (c^\top x' - c^\top x^0),\end{aligned}$$

where the first inequality holds because both x^0 and x' are 0-1 vectors and the second inequality results from $d^{*\top}x^0 \geqslant d^{*\top}x'$. This concludes the proof. □

This lower bound cannot be reached for all the inverse L_1 knapsack instances. Let us illustrate this point with two examples. Consider the following knapsack instance:

$$\begin{array}{rrcrcrcl} \max & f(x) = & 4x_1 & + & 5x_2 & + & 6x_3 & \\ \text{subject to:} & & x_1 & + & x_2 & + & x_3 & \leqslant 1\\ & & & & & & x_1, x_2, x_3 & \in \{0,1\}, \end{array}$$

where $x^* = (0,0,1)$ is an optimal solution. If the feasible solution $x^0 = (1,0,0)$ is chosen, then $(c^\top x^* - c^\top x^0) = 2$ is obtained. It is easy to see that $d^* = (6,5,6)$ with $|d^* - c|_1 = 2 = (c^\top x^* - c^\top x^0)$.

On the other hand, the following counter example shows this negative result:

$$\begin{array}{rrcrcrcl} \max & f(x) = & x_1 & + & x_2 & + & x_3 & \\ \text{subject to:} & & x_1 & + & x_2 & + & x_3 & \leqslant 1\\ & & & & & & x_1, x_2, x_3 & \in \{0,1\} \end{array}$$

If $x^0 = (0,0,0)$ is considered, then $(c^\top x^* - c^\top x^0) = 1$ is obtained. However, it is easy to see that the optimal solution of the IKP-1 must be $d^* = (0,0,0)$ with $|d^* - c|_1 = 3$. Consequently, the lower bound cannot be reached for all instances.

This leads us to the following lemma, which provides an upper bound on the optimal solution value of IKP-1.

Lemma 3. *Let $D_1 \in \mathbb{N}$ denote the optimal solution value of IKP-1. Then, $D_1 \leqslant \sum_{j \in J}\{(1 - x_j^0)c_j\}$.*

Proof. Proved by using the same arguments as in Lemma 1. □

Complexity Results

Let us analyze the complexity of IKP-1. We shall start by defining the corresponding decision problem.

The Inverse $\{0,1\}$-Knapsack Decision Problem under L_1 (IKDP-1)
INSTANCE: An instance of the $\{0,1\}$-knapsack problem (X, c), a feasible

solution $x^0 \in X$ and a $k \in \mathbb{N}^0$.
QUESTION: Is there a vector $d \in \mathbb{N}^n$ so that, $\sum_{j\in J} |c_j - d_j| \leqslant k$, $d^\top x^* = d^\top x^0$ and $x^* \in \arg\max\{d^\top x : x \in X\}$?

It is easy to verify that if IKDP-1 can be solved in polynomial time, then IKP-1 can also be solved in polynomial time by applying a binary search.

Theorem 5. *IKDP-1 is* ***co-NP****-Hard*

Proof. Consider the complement of the {0,1}-Knapsack decision problem denoted by $\overline{\text{KDP}}$ that is stated below.

INSTANCE: An instance of the $\{0,1\}$-knapsack problem (X, c) and a $t \in \mathbb{N}^0$.
QUESTION: Is the condition $\sum_{j\in J} x_j c_j < t$ fulfilled for all $x \in X$?

It is easy to see that $\overline{\text{KDP}}$ is the complement of KDP. Thus, by the **NP**-Completeness of KDP, one obtains that $\overline{\text{KDP}}$ is **co-NP**-Complete. The **co-NP**-Hardness of IKDP-1 is provided by a polynomial time reduction from $\overline{\text{KDP}}$ to IKDP-1. An instance of $\overline{\text{KDP}}$ is transformed into an instance of IKDP-1 by adding an item to the set J, with $c_{n+1} = t - 1$, $w_{n+1} = W$, $x_j^0 = 0$ for $j = 1, \dots, n$, $x_{n+1}^0 = 1$, and $k = 0$. The correctness of this reduction is as follows. For any given "Yes" instance of $\overline{\text{KDP}}$, for all $x \in X$ one obtains $\sum_{j\in J} x_j c_j < t$. Therefore, in the corresponding IKDP-1 instance the conditions $d^\top x^* = d^\top x^0$ and $x^* \in \arg\max\{d^\top x : x \in X\}$ are satisfied. For any "No" instance of $\overline{\text{KDP}}$, there exists an $x \in X$ such that $\sum_{j\in J} x_j c_j \geqslant t$. Therefore, in the corresponding IKDP-1 instance the conditions $d^\top x^* = d^\top x^0$ and $x^* \in \arg\max\{d^\top x : x \in X\}$ are not satisfied because $d^\top x^0 = t - 1$ and by hypothesis, there exists an $x \in X$ such that $d^\top x = t$. Consequently, IKDP-1 is **co-NP**-Hard and the theorem is proved. □

However, IKDP-1 has not been proved to belong to **co-NP** because the knapsack problem is required for building a certificate for the "No" instances. Indeed, IKDP-1 can be solved by an oracle Turing machine $\mathbf{NP}^{KDP}$ as proven below.

Theorem 6. *IKDP-1 belongs to* $\mathbf{NP}^{KDP}$

Proof. A nondeterministic polynomial time oracle Turing machine with KDP oracle can check whether $d \in \mathbb{N}^n$ satisfies the conditions $\sum_{j\in J} |c_j - d_j| \leq k$, $d^\top x^* = d^\top x^0$, and $x^* \in \arg\max\{d^\top x : x \in X\}$. The nondeterministic machine guesses vector d. Then, the first condition is checked in polynomial time and the two last are checked by using the KDP oracle. □

From this complexity analysis, one can conclude that the problem should be harder to solve than IKP-∞ and that there is less hope of finding a pseudo-polynomial time algorithm to solve it. This is why the use of integer linear programing and bilevel integer linear programing models are investigated in the remaining of this chapter.

An Integer Linear Programming Formulation

An integer linear program for computing d^* is proposed here. Let us recall the classical dynamic programming approach for a given instance (X, d) of the {0,1}-Knapsack problem. Let $g_i(q)$ denote the maximum profit achievable when considering the first i items of J, with $i \in J$ and a capacity $q \in \{0, 1, \ldots, W\}$. The value of $g_i(q)$ can be determined through the following linear $\{0,1\}$ model.

$$\begin{array}{rll} g_i(q) = \max & \sum_{j=1}^{i} d_j x_j & \\ \text{subject to:} & \sum_{j=1}^{i} w_j x_j & \leqslant q \\ & x_j & \in \{0,1\},\ j \in \{1, \ldots, i\} \end{array}$$

Note that the original knapsack problem is to find $g_n(W)$. It is widely known that the following recursive formula solves the knapsack problem (see for example Kellerer et al. 1994).

$$g_i(q) = \begin{cases} g_{i-1}(q), & \text{if } q < w_i, \\ \\ \max\left\{g_{i-1}(q - w_i) + d_i, g_{i-1}(q)\right\}, & \text{if } q \geqslant w_i. \end{cases}$$

with $g_1(q) \leftarrow 0$ for $q = 0, 1, \ldots, w_1 - 1$ and $g_1(q) \leftarrow d_1$ for $q = w_1, w_1 + 1, \ldots, W$. This approach can be modeled as an integer linear program. Consider the following set of constraints denoted by (6.2).

$$\begin{cases} g_1(q) \geqslant 0, & \text{for all } q = 0, 1, \ldots, w_1 - 1, \\ g_1(q) = d_1, & \text{for all } q = w_1, w_1 + 1, \ldots, W, \\ g_i(q) = g_{i-1}(q), & \text{for all } q = 0, 1, \ldots, w_i - 1, \text{ and } i \in \{j \in J : j \geqslant 2\}, \\ g_i(q) \geqslant g_{i-1}(q), & \text{for all } q = w_i, 1, \ldots, W, \text{ and } i \in \{j \in J : j \geqslant 2\}, \\ g_i(q) \geqslant g_{i-1}(q - w_i) + d_i, & \text{for all } q = w_i, w_i + 1, \ldots, W, \text{ and } i \in \{j \in J : j \geqslant 2\}. \end{cases} \tag{6.2}$$

By minimizing $g_n(W)$ over this set of constraints, it is easy to see that one obtains the optimal value of the knapsack problem. Consider the following integer linear programming model (Problem 6.3).

$$\begin{array}{rl} \min & \sum_{j \in J} \delta_j \\ \text{subject to:} & \delta_j \geqslant d_j - c_j,\ j \in J \\ & \delta_j \geqslant c_j - d_j,\ j \in J \\ & \sum_{j \in J} d_j x_j^0 \geqslant g_n(W) \\ & \text{The set of constraints (6.2) on } g_n(W) \\ & d \in \mathbb{N}^n \\ & \delta_j \in \mathbb{N},\ j \in J \end{array} \tag{6.3}$$

Similar to the dynamic programming algorithm for the knapsack problem, this integer program can be built with an algorithm that runs in $O(nW)$.

The following theorem establishes the exactness of this formulation for solving IKP-1.

Theorem 7. *Problem 6.3 determines a vector* $d^* \in \mathbb{N}^n$ *with* $d^{*\top}x^0 = \max\{d^{*\top}x : x \in X\}$ *and such that there is no* $d' \in \mathbb{N}^n$ *with* $\sum_{j \in J} |c_j - d'_j| < \sum_{j \in J} |c_j - d^*_j|$ *and* $d'x^0 = \max\{d'x : x \in X\}$.

Proof. The feasible solution x^0 is optimal for all vectors $d \in \mathbb{N}^n$ satisfying the set of constraints described in Problem 6.3. Indeed, $g_n(W) \geqslant \max\{d^\top x : x \in X\}$ and the set of constraints implies $d^\top x^0 \geqslant g_n(W)$. Furthermore, all vectors leading to the optimality of x^0 satisfy the set of constraints. Indeed, for all vectors $d \in \mathbb{N}^n$ with $d^\top x^0 = \max\{d^\top x : x \in X\}$, one can build a $g_n(W) \in \mathbb{N}$ such that $\sum_{j \in J} d_j x^0_j = g_n(W)$. This results from the definition of $g_n(W)$. This concludes the proof.

□

If the integrality constraint on vector d is removed, Problem 6.3 becomes a linear programming model. Then, Problem 6.3 can be solved through a pseudo-polynomial time algorithm. Indeed, it is known that Karmarkar's algorithm for solving linear programming runs in $O(n^{3,5}L)$, where n denotes the number of variables and L the number of bits in the input (Karmarkar 1984). Problem 6.3 is composed of $O(nW)$ variables. Therefore, one can build an algorithm for computing $d^* \in \mathbb{R}^n$ that runs in $O\Big((nW)^{3,5} log(nW)\Big)$. Of course, this is a theoretical result and Karmarkar's algorithm should be replaced in practice by the simplex algorithm.

6.4 A Bilevel Programming Approach

Since the inverse $\{0,1\}$-knapsack can be defined as a bilevel problem, a natural approach for solving this problem is to consider bilevel programming techniques. This section presents how these techniques can be applied to the inverse $\{0,1\}$-knapsack under the L_∞ norm. However, it is easy to extend this approach to the L_1 norm.

Linearization of the Inverse $\{0,1\}$-Knapsack Problem

This section deals with the linearization of the bilevel problem IKP-∞. We shall start by introducing a variable $\delta \in \mathbb{R}$ to remove the maximization operator. The problem can be stated as follows:

$$\begin{array}{rl} \min & \delta \\ \text{subject to:} & \delta \geqslant c_j - d_j, \ j \in J \\ & \delta \geqslant d_j - c_j, \ j \in J \\ & d^\top x^* = d^\top x^0 \\ & x^* \in \arg\max\{d^\top x : x \in X\} \\ & d \in \mathbb{N}^n \\ & \delta \in \mathbb{R} \end{array}$$

The non-linear vector product $d^\top x^*$ can be replaced by a sum of n real-valued variables. Create n new variables $e_j \in \mathbb{R}_+$ with $j \in J$ that are equal either to zero or bounded from above by d_j. This is expressed by the two following constraints for each variable e_j with $j \in J$.

$$\begin{cases} e_j \leqslant d_j, \\ \\ e_j \leqslant x_j M. \end{cases}$$

where $M = 2\max_{j \in J}\{c_j\}$ is the upper bound on the the value of d_j^*, with $j \in J$. This upper bound is directly deduced from Lemma 1.

Finally, replace $d^\top x$ with $\sum_{j\in J} e_j$. The resulting bilevel integer linear programming version of the inverse $\{0,1\}$-knapsack problem under the L_∞ norm can be stated as follows:

$$\begin{array}{rll} \min & \delta & \\ \text{subject to:} & \delta \geqslant d_j - c_j,\ j \in J & \\ & \delta \geqslant c_j - d_j,\ j \in J & \\ & \sum_{j\in J} e_j^* = d^\top x^0 & \\ & d \in \mathbb{N}^n & \\ & \delta \in \mathbb{R} & \\ & e^* \in \arg\max \sum_{j\in J} e_j & \\ & \qquad\qquad \text{subject to:} & e_j \leqslant d_j,\ j \in J \\ & & e_j \leqslant x_j M,\ j \in J \\ & & e \in \mathbb{R}^n_+ \\ & & x \in X \end{array} \tag{6.4}$$

Now, let us prove the correctness of this formulation. Consider the two following lemmas establishing the optimality of e^*, that is $\sum_{j\in J} e_j^* = \max\{d^\top x : x \in X\}$.

Lemma 4. *If $e \in \mathbb{R}^n_+$ and $x \in X$ denote a feasible solution of the lower-level problem of the bilevel optimization problem (6.4), then $\sum_{j\in J} e_j \leqslant \sum_{j\in J} x_j d_j$.*

Proof. The two following relations between $e \in \mathbb{R}^n_+$ and $x \in X$ are directly deduced from the set of constraints of the lower-level problem.

$$\begin{cases} (e_j > 0) \Rightarrow (x_j = 1), \\ \\ (e_j = 0) \Rightarrow (x_j = 1) \vee (x_j = 0). \end{cases}$$

Due to the set of constraints of the lower-level problem, one obtains $e_j \leqslant d_j$. Therefore, $e_j \leqslant x_j d_j$ for all $j \in J$, which implies $\sum_{j\in J} e_j \leqslant \sum_{j\in J} x_j d_j$. This concludes the proof. □

Lemma 5. *For all feasible solutions $x \in X$, there exists a vector $e \in \mathbb{R}^n_+$ satisfying the set of constraints of the lower-level problem of the bilevel optimization problem (6.4) and such that $\sum_{j\in J} e_j = \sum_{j\in J} x_j d_j$.*

Proof. Define the vector $e \in \mathbb{R}^n_+$ as follows. For all $j \in J$:

$$\begin{cases} e_j = d_j, & \text{if } x_j = 1, \\ e_j = 0, & \text{otherwise.} \end{cases}$$

this implies that $\sum_{j \in J} e_j = \sum_{j \in J} x_j d_j$. Furthermore, it is easy to see that vector e satisfies the set of constraints. This concludes the proof. □

Theorem 8. *Let $e^* \in \mathbb{R}^n_+$ denote an optimal solution of the lower level problem of the bilevel optimization problem (6.4). Then, $\sum_{j \in J} e^*_j = \max\{d^\top x : x \in X\}$.*

Proof. This results directly from Lemmas 4 and 5. □

Analysis of the Bilevel Integer Linear Programming Problem

Consider the use of bilevel integer linear programming for solving IKP-∞. The existing methods for solving bilevel integer linear programming problems do not allow to have constraints on variables e_j in the upper level problem (Moore and Bard 1990, DeNegre and Ralphs 2009). However, this can be easily tackled by inserting the constraint $\sum_{j \in J} e_j = d^\top x^0$ into the objective function. With $L = C + 1$, the following equivalent problem is obtained.

$$\begin{array}{rll} \min & \delta + \Big(\sum_{j \in J} e_j - d^\top x^0\Big) L & \\ \text{subject to:} & \delta \geqslant d_j - c_j, \; j \in J & \\ & \delta \geqslant c_j - d_j, \; j \in J & \\ & d \in \mathbb{N}^n & \\ & \delta \in \mathbb{R} & \\ & e \in \arg\max \sum_{j \in J} e_j & \\ & \qquad \text{subject to:} & e_j \leqslant d_j, \; j \in J \\ & & e_j \leqslant x_j M, \; j \in J \\ & & e \in \mathbb{R}^n_+ \\ & & x \in X \end{array} \tag{6.5}$$

Based on Lemma 1, it is known that $\delta \leqslant C$. Therefore, a solution of Problem (6.5) with $d^\top x^0 < \sum_{j \in J} e_j$ is not an optimum, because in this case, $\delta + \Big(\sum_{j \in J} e_j - d^\top x^0\Big) L \geqslant L > C$.

This integer bilevel linear programming problem can be solved by a branch-and-cut algorithm (DeNegre and Ralphs 2009) or by a branch-and-bound algorithm (Moore and Bard 1990).

6.5 Computational Experiments

The purpose of this section is to report the behavior of Algorithm 7 and the integer programming model of Problem 6.3 on several sets of randomly generated instances. This helps to measure the practical application in terms of performance of the approaches proposed in this paper. The design of the experiments is inspired by the frameworks used in Martello and Toth (1990) and in Pisinger (1995). Experiments on the bilevel model are not included due to the poor performance of the existing procedures (DeNegre and Ralphs 2009, Moore and Bard 1990).

Design of the Experiments

For a better understanding of how the methods behave, several groups of randomly generated instances of the {0,1}-knapsack problem were considered. For a given number of variables n and *data range* R, instances were randomly generated in three different ways:

- **Uncorrelated instances** where c_j and w_j are randomly distributed in $[1, R]$;
- **Weakly correlated instances** where w_j is randomly distributed in $[1, R]$ and c_j is randomly distributed in $[w_j - R/10, w_j + R/10]$ such that $c_j \geqslant 1$;
- **Strongly correlated instances** where w_j is randomly distributed in $[1, R]$ and $c_j = w_j + 10$.

The reader should refer to Kellerer et al. (1994) for more detailed information on the instances.
For each class of instance, three types of groups were constructed:

- **Type 1** where each instance $i \in \{1, \ldots, S\}$ is generated with the seed number i and W is computed as the maximum between R and $\lfloor i/(S+1) \sum_{j \in J} w_j \rfloor$;
- **Type 2** where each instance $i \in \{1, \ldots, S\}$ is generated with the seed number i and W is computed as the maximum between R and $\lfloor P \sum_{j \in J} w_j \rfloor$, where $P \in [0, 1]$;
- **Type 3** where each instance $i \in \{1, \ldots, S\}$ is generated with the seed number $i \cdot 10$ and W is computed as the maximum between R and $\lfloor P \sum_{j \in J} w_j \rfloor$, where $P \in [0, 1]$.

In the three cases, each instance is completed by a feasible solution x^0 defined through the traditional greedy heuristic algorithm (see Kellerer et al. 1994). This provides a feasible solution that is sufficiently close to the optimal one. Two random generators were used to build these instances; the one used in the HP9000 - UNIX and the random function of the NETGEN generator (Klingman et al. 1974).

Groups of Type 1 were composed by $S = 100$ instances as proposed in Kellerer et al. (1994) and groups of Type 2 and 3 were composed by $S = 30$

instances. The choice of 30 is based on the *rule of thumb* in statistics to produce good estimates (Coffin and Saltzman 2000).

For each group of instances, the performance of the approaches were measured through the average (Avg.), standard deviation (Std. dev.), minimum (Min.) and maximum (Max.) of the CPU time in seconds.

Algorithm 7 was implemented in the C++ programming language and integrates the algorithm for the {0,1}-knapsack problem proposed by Pisinger (1997)[1] and Problem 6.3 is solved by using the Cplex solver. All the experiments were performed on a multiple processors architecture composed of four 2.8 GHz AMD Opteron dual-core processors, 32GB of RAM, and running Linux as the operating system. This makes it possible to take advantage of the parallel algorithms found in the Cplex solver. A limit of 10 hours was assigned to each group of S instances. An excess time limit is represented by a dash in the tables.

Statistical Analysis

For the analysis of Algorithm 7, only strongly correlated instances are considered as they appear as being the hardest instances for the knapsack solver used in both algorithms. Furthermore, only the results on Type 2 instances generated by the HP-9000 UNIX random function are presented in this section as they give rise to the same conclusions as the other types of randomly generated instances. For more details on the results of the computational experiments, the reader may consult Appendix A.

Algorithm 7 is very efficient for large scale instances. For example, with $n = 10^5$, $R = 10^4$ and $P = 0.5$, the average computation time is 27.37 seconds with a 95% confidence interval $[17.96, 36.79]$ obtained by using the Student's t-Distribution with 29 degrees of freedom. Let us point out that the performances of this algorithm are strongly linked to the embedded knapsack solver. Therefore, the use of another solver could either increase or decrease significantly the computation time.

For the experiments on the integer linear programming formulation for solving IKP-1 we shall also present the results for the uncorrelated instances because of the hardness of this problem.

When considering small uncorrelated instances, the inverse problem can be solved with a reasonable computation time. For example, with $n = 50$, $R = 500$, and $P = 0.5$, the average computation time is 79.41 seconds with a 95% confidence interval $[50.95, 107.88]$. However, by the nature of the formulation, the computation time becomes quickly unreasonable. For example, with $n = 80$, $R = 500$, and $P = 0.5$, the average computation time is 744.64 seconds with a 95% confidence interval $[430.58, 1058.71]$.

The use of strongly correlated instances has a strong impact on the performance. For example, with $n = 50$, $R = 100$, and $P = 0.5$, the average

---

[1] The implementation of this algorithm is available at `http://www.diku.dk/~pisinger/codes.html`

computation time for the strongly correlated instances is 209.38 seconds with a 95% confidence interval $[150.09, 268.67]$, while for the uncorrelated instances the average computation time is 6.45 seconds with a 95% confidence interval $[3.93, 8.97]$.

6.6 Conclusion

In this chapter, the inverse knapsack problem has been defined and studied for both the L_∞ and L_1 norms. Complexity analysis has highlighted the theoretical hardness of both problems. Despite the hardness of IKP-∞, experimental results have shown the tractability of the computational procedure used for solving it. On the other hand, the method proposed here for solving IKP-1 can only handle small instances. However, this result is partially explained by the **co-NP**-Hardness of this problem. Therefore, the use of approximation algorithms should be considered in the future.

There are many possible extensions of this work. For instance, the use of other norms, the nature of the adjustment, and constraints on the adjustment are of obvious interest. Furthermore, this work clears the way for solving the inverse version of other optimization problems such as the multi-constraint knapsack problem and the integer knapsack problem. Indeed, some of the proposed approaches could be easily extended to these problems.

The knowledge acquired in this chapter is a significant help for solving some inverse multi-objective combinatorial optimization problems. Indeed, in the next chapter several algorithms will rely on the same observations that the ones we made in this chapter to solve the inverse knapsack.

Chapter 7

Inverse multi-objective optimization

In this chapter, inverse optimization is extended to multi-objective optimization. Inverse multi-objective combinatorial optimization is firstly defined as the problem of finding a minimal adjustment of the objective functions coefficients such that a given feasible solution becomes an efficient one. A more general definition is then considered. It is stated as the problem of finding a minimal adjustment of the optimization instance inducing a change in the efficient set and/or in the non-dominated set. This definition raises many questions that are formalized by a triplet which highlights a large collection of inverse problems that could be investigated. Several inverse problems are presented along with their respective applications. Finally, a collection of algorithms to solve several inverse problems are presented.

7.1 Introduction

The purpose of this chapter is to address the question of extending inverse optimization to the context of multi-objective optimization. As explained in the introduction this constitutes our main focus. To our knowledge this question has never been addressed in previous works.

Let us start with a first definition of inverse multi-objective optimization. This definition is a particular case of a more general definition that will be stated afterward. For this purpose, we consider the most usual definition of inverse optimization in single-objective optimization. It consists of finding a minimal adjustment of the coefficients of the objective function (*i.e.*, the profit vector) such that a given solution becomes optimal. In the context of multiple objective functions, the adjustment should be performed on the coefficients of all the objective functions (*i.e.*, the profit matrix), which can be measured by any distance function between matrices. As explained in Chapter 4, there is usually no feasible ideal solution, but a set of efficient solutions.

Consequently, the concept of "optimality" could be replaced by "efficiency". Therefore, the most straightforward extension of inverse optimization to multi-objective optimization problems consists of finding a minimal adjustment of the profit matrix such that a given feasible solution becomes efficient.

This question has potential theoretical and practical implications. It leads to a way for measuring "the distance to efficiency" for a feasible solution. This distance is equal to zero if and only if the solution is efficient, and increases with how much the coefficients have to be modified so that the solution becomes efficient. An illustrative application of inverse optimization in the field of multi-objective optimization drawn from portfolio analysis can be stated as follows. Portfolios are generally built by searching the ones that minimize risk and simultaneously maximize return, dividends, liquidity, etc. Given a portfolio, it is nevertheless difficult to evaluate its performances on this set of objectives. In such a kind of problems it is common to model the expected return by the average daily return. However, when looking about the different periods to observe, there is no *consensus* among researchers and practitioners. It might be calculated over the last week, the last month, etc. We could start to build a general model in order to identify efficient solutions. An observed portfolio might be dominated in this model because investors *perceive* or *evaluate* parameters in a slightly different way. However, a natural constraint is to impose the efficiency of the observed portfolios. Therefore, one could focus on the minimal adjustment of the initial model parameters in order to satisfy this constraint. In a first attempt for solving this problem, we could assume that all the objectives and constraints are linear. Within these assumptions, the model leads to an instance of the inverse multi-objective knapsack problem, *i.e.*, given an instance and a set of feasible solutions, the question is how to modify the profits as little as possible such that those solutions become efficient.

7.2 Inverse problems

As explained in Chapter 5, inverse optimization can be more broadly defined as the problem of finding a minimal adjustment of an optimization instance (X, c) inducing a change in the optimal solution's characteristics. Hence, a more general definition of inverse multi-objective optimization can be stated as follows.

Definition 26 (inverse multi-objective optimization)**.** *Given an instance* (X, C)*, an inverse problem consists of finding a minimal adjustment of* (X, C) *inducing a change in the efficient set* $E(X, C)$ *and/or in the non-dominated set* $ND(X, C)$*.*

This definition raises many questions that can be formalized as follows. Let us consider an instance (X, C). Inverse multi-objective optimization consist of finding another instance $(\mathcal{X}^*, D^*)$ that satisfies some conditions and such that the distance to (X, C) is minimized. A feasible solution to an inverse problem is denoted by $(\mathcal{X}, D)$ and an optimal solution is denoted by $(\mathcal{X}^*, D^*)$. In order to not get lost in all these questions, we describe each inverse multi-objective optimization problem by a triplet $[\alpha \bullet \beta \bullet \gamma]$. The α field describes how the instance is allowed to be modified, *i.e.*, a set of conditions on $(\mathcal{X}, D)$. The β field describes the conditions on $ND(\mathcal{X}, D)$ and/or $E(\mathcal{X}, D)$. The γ field describes how the modification is measured.

There are many possible ways to instantiate these fields and therefore many possible inverse multi-objective optimization problems could be defined. For example, the α field may be composed of elements from the set $\{D \in \mathbb{N}^{q\times n}, D = C, \mathcal{X} \subseteq \{x : x \in \{0,1\}^n\}, \mathcal{X} = X, X \subseteq \mathcal{X}, \ldots\}$. The β field may be composed of elements from the set $\{E(\mathcal{X}, D) \subseteq E(X, C), x^0 \in E(\mathcal{X}, D), X^0 \subseteq E(\mathcal{X}, D), \ldots\}$, where $x^0 \in X$ and $X^0 \subseteq X$ are given as input to the inverse problem. The γ fields may be composed of elements from the set $\{|C - D|_p, ||C - D||_p, \delta(C, D), \ldots\}$, where $p \geqslant 1$.

Let us illustrate this nomenclature with the following examples. In these examples, the feasible set X is not modified, *i.e*, $\mathcal{X}$ is equal to X. Therefore, the β field describes the conditions on $ND(X, D)$ and/or $E(X, D)$.

Example 1 (Distance to efficiency). $[D \in \mathbb{N}^{q\times n} \bullet x^0 \in E(X, D) \bullet |C - D|_p]$ *denotes the problem of computing the distance to efficiency of* $x^0 \in X$. *Let* D^* *denote an optimal solution to it. This inverse problem consists of finding a profit matrix* D^* *that is as close as possible to* C *such that* x^0 *is efficient, with respect to the distance function* $|C - D^*|_p$. *This distance is equal to zero when* $x^0 \in E(X, C)$ *and proportional with how much the profit matrix* C *must be modified to transform* x^0 *into an efficient solution. Hence, this is a way to measure the distance to the efficiency that might be seen as the quality of a solution.*

Example 2 (Stability radius). *The stability radius of an efficient solution is defined as the maximal variation of the problem parameters that allows this solution to remain efficient (see Emelichev et al. 2004, Emelichev and Kuzmin 2006, Emelichev and Podkopaev 2010). It is easy to see that the stability radius of an efficient solution can be obtained through the minimal adjustment of the parameters, in such a way that the solution becomes non-efficient. The inverse problem* $[D \in \mathbb{N}^{q\times n} \bullet x^0 \notin E(X, D) \bullet |C - D|_p]$ *can be used to answer this question, where only the profit matrix is modified. Note that the* α *field could also be defined by* $D \in \mathbb{N}^{q\times n}, \mathcal{X} \subseteq \{0,1\}^n$, *in which case all the problem parameters are allowed to be modified. This example is addressed in more details in Chapter 8.*

Example 3 (Stability of an efficient set). *The stability radius of an efficient solution can be generalized to the whole efficient set. This can be defined as the maximal variation of the problem parameters so that the efficient set remains unchanged. The inverse problem* $[D \in \mathbb{N}^{q\times n} \bullet E(X, C) \neq E(X, D) \bullet |C - D|_p]$ *can be used to compute it. This consists in finding the minimal adjustment of the profit matrix inducing a change (addition or removal of solutions in the efficient set).*

Distance function	Simplified form
$\lvert C - D\rvert_\infty$	$\lvert\cdot\rvert_\infty$
$\lvert C - D\rvert_1$	$\lvert\cdot\rvert_1$
Any linearizable distance function	$\delta(C, D)$

Table 7.1: Correspondence between distance functions and their simplified form

Example 4 (Distance to ideal - Compromise solution)**.** *$[D \in \mathbb{N}^{q\times n} \bullet \{Dx^0\} = ND(X, D) \bullet |C - D|_p]$ denotes the problem of computing the distance to ideal of $x^0 \in X$. It consists of finding a profit matrix D^* that is as close as possible to C such that the vector D^*x^0 is the ideal outcome. As it will be explained in Chapter 8, this inverse problem leads to a concept of compromise solution in the context of combinatorial optimization problems with multiple experts.*

From our perspective, these examples provide a significant motivation to the study of inverse multi-objective optimization problems. In the remaining sections of this chapter we will focus on a theoretical exploration of inverse problems. The stability radius and the concept of compromise solutions are discussed in more details in Chapter 8.

The notation $[\alpha\bullet\beta\bullet\gamma]$ may easily become cumbersome. It can be simplified by omitting the α and the γ fields. This leads to the inverse problems denoted by $[\bullet\beta\bullet]$, $[\bullet\beta\bullet\gamma]$, or $[\alpha\bullet\beta\bullet]$. If the α field is omitted, then it takes the default value of $D \in \mathbb{N}^{q\times n}$. If the γ field is omitted, then the adjustment is measured by any linearizable distance function between C and D. Furthermore, the γ field can be simplified if only matrices C and D are compared. Table 7.2 gives the correspondence between the distance functions and their respective simplifications.

In the following sections are presented a collection of algorithms to solve several inverse problems. We selected these inverse problems due to their potential applications. Indeed, some of these algorithms will be used in the next chapter for the computation of stability radii and compromise solutions.

7.3 A binary search algorithm for $[\bullet x^0 \in E(X, D) \bullet |\cdot|_\infty]$

The inverse problem $[\bullet x^0 \in E(X, D) \bullet |\cdot|_\infty]$ consists of finding a minimal adjustment of the profit matrix, measured by the Chebyshev norm, such that a given feasible solution becomes efficient. The inverse problem can be stated as follows:

$$\begin{aligned} D^* \in \arg\min \quad & |C - D|_\infty \\ \text{subject to:} \quad & x^0 \in E(X, D) \\ & D \in \mathbb{N}^{q\times n}. \end{aligned} \qquad ([\bullet x^0 \in E(X, D) \bullet |\cdot|_\infty])$$

This problem can be modeled as the following bilevel optimization problem:

$$\begin{array}{rll} D^* \in \arg\min & |C-D|_\infty & \\ \text{subject to:} & Dx^0 = Dx^* & \\ & D \in \mathbb{N}^{q\times n} & \\ & x^* \in \arg\max & \sum_{i\in I}\sum_{j\in J} D_{ij}x_j \\ & \text{subject to:} & Dx \geqq Dx^0 \\ & & x \in X \end{array} \tag{7.1}$$

where we seek for a profit matrix $D^* \in \mathbb{N}^{q\times n}$, which minimizes the L_∞ distance with respect to C and such that x^0 is an efficient solution of the modified optimization problem (X, D^*). The efficiency of x^0 is ensured by Problem ND-Test (see, Section 4.1) as the second level optimization problem.

Theoretical results

At first, let us analyze the nature of some optimal solutions of $[\bullet x^0 \in E(X,D)\bullet |\cdot|_\infty]$. Based on a partition of J defined by $J^0 = \{j \in J : x^0_j = 0\}$ and $J^1 = \{j \in J : x^0_j = 1\}$, the first theorem establishes that an optimal solution D^* can be built by increasing, or keeping equal, C_{ij}, for all $j \in J^1$ and by decreasing, or keeping equal, C_{ij}, for all $j \in J^0$, for $i \in I$.

Theorem 9. *There exists an optimal solution $D^* \in \mathbb{N}^{q\times n}$ of $[\bullet x^0 \in E(X,D)\bullet |\cdot|_\infty]$ such that $\forall j \in J^1 : D^*_{ij} \geqslant C_{ij}$ and $\forall j \in J^0 : D^*_{ij} \leqslant C_{ij}$, with $i \in I$.*

Proof. Let $D \in \mathbb{N}^{q\times n}$ denote any optimal solution of $[\bullet x^0 \in E(X,D)\bullet |\cdot|_\infty]$. Define the following sets for all $i \in I : J^{0>}_i = \{j \in J^0 : D_{ij} > C_{ij}\}$, $J^{1<}_i = \{j \in J^1 : D_{ij} < C_{ij}\}$. Consider a solution D^* of $[\bullet x^0 \in E(X,D)\bullet |\cdot|_\infty]$ defined as follows, for all $i \in I$:

$$D^*_{ij} = \begin{cases} C_{ij}, & \text{if } j \in \{J^{1<}_i \cup J^{0>}_i\} \\ \\ D_{ij}, & \text{otherwise.} \end{cases} \tag{7.2}$$

Let us show that for all $x \in X$, if $Dx^0 \geq Dx$, then $D^*x^0 \geq D^*x$. This is equivalent to show that the following condition holds: if $D(x^0 - x) \geq 0$, then $D^*(x^0 - x) \geq 0$. The profit matrix D^* is introduced in the first inequality as follows, $(D - D^* + D^*)(x^0 - x) \geq 0$, which leads to write: $D^*(x^0 - x) \geq (D^* - D)(x^0 - x)$. From Equation 7.2 and the definition of J^1, one may deduce that, for all $j \in J^1$, $(x^0 - x)_j \geqslant 0$ and $(D^*_{ij} - D_{ij}) \geqslant 0$. Similarly, for all $j \in J^0$, $(x^0 - x)_j \leqslant 0$ and $(D^*_{ij} - D_{ij}) \leqslant 0$. Therefore, $(D^* - D)(x^0 - x) \geqq 0$, and consequently $D^*(x^0 - x) \geq 0$. □

Let us define a matrix $D^k \in \mathbb{N}^{q\times n}$ of distance at most k from matrix C with respect to the L_∞ norm.

Definition 27 (D^k)**.** *Let $k \geqslant 0$ be a natural number. Then, D^k is a matrix of size $q \times n$, where for all $i \in I$, and $j \in J$,*

$$D^k_{ij} = \begin{cases} \max\{0, C_{ij} - k\}, & \text{if } x^0_j = 0, \\ \\ C_{ij} + k, & \text{otherwise.} \end{cases}$$

Theorem 10. *If* D^* *represents an optimal solution of* $[\bullet x^0 \in E(X,D) \bullet |\cdot|_\infty]$*, such that* $|C-D|_\infty = k$*, then* D^k *is also an optimal solution of* $[\bullet x^0 \in E(X,D) \bullet |\cdot|_\infty]$*.*

Proof. Let D^* denote an optimal solution of $[\bullet x^0 \in E(X,D) \bullet |\cdot|_\infty]$, such that $|C-D|_\infty = k$. Let us show that for all $x \in X$, if $D^* x^0 \geq Dx$, then $D^k x^0 \geq D^k x$. Similarly to the proof of Theorem 9, this is equivalent to show that the following condition holds: if $D^*(x^0 - x) \geq 0$, then $D^k(x^0 - x) \geq 0$. This leads to write: $D^k(x^0 - x) \geq (D^k - D^*)(x^0 - x)$. From Definition 27 and the definition of J^0, one may deduce that, for all $j \in J^1$, $(x^0 - x)_j \leqslant 0$ and $(D^k_{ij} - D^*_{ij}) \leqslant 0$. Similarly, for all $j \in J^1$, $(x^0 - x)_j \geqslant 0$ and $(D^k_{ij} - D^*_{ij}) \geqslant 0$. Therefore, $(D^k - D^*)(x^0 - x) \geqq 0$, and consequently $D^k(x^0 - x) \geq 0$.

$\square$

Lemma 6. *If* $\delta_\infty \in \mathbb{N}$ *is the optimal solution value of* $[\bullet x^0 \in E(X,D) \bullet |\cdot|_\infty]$*, then* $\delta_\infty \leqslant \Delta = \max_{i \in I, j \in J^0}\{C_{ij}\}$

Proof. It is always possible to build a matrix $D \in \mathbb{N}^{q \times n}$, with $|C - D|_\infty = \max_{i \in I, j \in J^0}\{C_{ij}\}$, such that Dx^0 is a non-dominated solution of (X, D). The matrix is defined as follows, $\forall i \in I, \forall j \in J^1 : D_{ij} = C_{ij}$ and $\forall i \in I, \forall j \in J^0 : D_{ij} = 0$. It is easy to see that for all $x \in X$, one obtains $Dx^0 \geqq Dx$ and $|C - D|_\infty = \max_{i \in I, j \in J^0}\{C_{ij}\}$. This concludes the proof. $\square$

Lemma 7. *If* $D^k x^0$ *is a non-dominated vector for* (X, D^k)*, then* $D^{k+1} x^0$ *is a non-dominated vector for* (X, D^{k+1})*.*

Proof. Let us assume that there exists a solution $x \in X$ such that $D^{k+1}x$ dominates $D^{k+1}x^0$. Therefore, $D^{k+1}(x - x^0) \geq 0$. The profit matrix D^k is introduced in the inequality as follows, $(D^{k+1} + D^k - D^k)(x - x^0) \geq 0$, which leads to write: $D^k(x - x^0) \geq (D^k - D^{k+1})(x - x^0)$. From Definition 27 and the definition of J^1, one may deduce that, for all $j \in J^1$, $(x - x^0)_j \leqslant 0$ and $(D^k_{ij} - D^{k+1}_{ij}) \leqslant 0$. Similarly, for all $j \in J^0$, $(x - x^0)_j \geqslant 0$ and $(D^k_{ij} - D^{k+1}_{ij}) \geqslant 0$. Therefore, $(D^k - D^{k+1})(x - x^0) \geqq 0$, and consequently $D^k(x - x^0) \geq 0$. Therefore, $D^k x$ dominates $D^k x^0$ which contradicts the hypothesis, and the lemma is proved. $\square$

Algorithm

Based on the results presented in the previous section, a binary search algorithm for computing an optimal solution of $[\bullet x^0 \in E(X,D) \bullet |\cdot|_\infty]$ is devised. Thanks to Theorem 10, an optimal solution of $[\bullet x^0 \in E(X,D) \bullet |\cdot|_\infty]$ can be built based on the distance between matrices D^* and C. Since this distance is bounded from above by $\Delta = \max_{i \in I, j \in J}\{(1 - x^0_j)C_{ij}\}$ (see Lemma 6), the algorithm consists of finding the minimal value $k \in \{1, 2, \ldots, \Delta\}$ such that $D^k x^0$ is a non-dominated vector of the MOCO instance (X, D^k). This condition is naturally checked by solving Problem ND-Test. The minimal value of k can be found by performing a binary search on the set $\{1, 2, \ldots, \Delta\}$ as a consequence of Lemma 7. Therefore, Algorithm 8 requires to solve $O(log_2 \Delta)$ times Problem ND-Test. For more details on the procedure, see the pseudo-code of Algorithm 8.

Algorithm 8 A binary search algorithm to compute an optimal solution D^* of $[\bullet x^0 \in E(X,D) \bullet \mid \cdot \mid_\infty]$, with respect to the Chebyshev norm.

```
$a \leftarrow 0$;
$b \leftarrow \Delta$;
while ($a \neq b$) do
    $k \leftarrow a + \lfloor (b-a)/2 \rfloor$;
    Build matrix $D^k$;
    $x^* \leftarrow$ Solve Problem ND-Test;
    if ($D^k x^0 = D^k x^*$) then
        $b \leftarrow k$;
    else
        $a \leftarrow k + 1$;
    end if
end while
$D^* \leftarrow D^a$;
```

The computational complexity of Algorithm 8 could be potentially improved by taking into account the characteristics of the combinatorial problem. Let us consider the multi-objective knapsack problem as an illustrative example. As presented in Chapter 4, a dynamic programming approach solves this problem in $O(n(UB^{\max})^q)$, where $UB^{\max} = \max\{UB_1, UB_2, \ldots, UB_i, \ldots, UB_q\}$ and UB_i is an upper bound on the optimal solution value of the *i-th* objective function. Therefore, in the context of $[\bullet x^0 \in E(X,D) \bullet \mid \cdot \mid_\infty]$, the inverse multi-objective knapsack problem can be solved in $O(log_2 \Delta n (UB^{\max})^q)$.

7.4 A binary search algorithm for $[\bullet x^0 \notin E(X,D) \bullet \mid \cdot \mid_\infty]$

The inverse problem $[\bullet x^0 \notin E(X,D) \bullet \mid \cdot \mid_\infty]$ consists of finding a minimal adjustment of the profit matrix such that a given feasible solution becomes non-efficient. This question is formalized as follows. Let (X, C) denote an instance of a MOCO and $x^0 \in X$ a feasible solution. The inverse problem can be stated as follows:

$$\begin{array}{rl} D^* \in \arg\min & \mid C - D \mid_\infty \\ \text{subject to:} & x^0 \notin E(X,D) \\ & D \in \mathbb{N}^{q \times n}. \end{array} \qquad ([\bullet x^0 \notin E(X,D) \bullet \mid \cdot \mid_\infty])$$

Let us first note that this inverse problem is not feasible whenever every solution $x \in X \backslash \{x^0\}$ is a subset of x^0, *i.e.*, when there does not exists a $j \in J$

such that $x_j = 1$ and $x_j^0 = 0$. This situation occurs, for example, in the following instance:

$$\begin{array}{rrrrrl} \max & f_1(x) = & 5x_1 & + & 5x_2 & \\ \max & f_2(x) = & 3x_1 & + & 2x_2 & \\ \text{subject to:} & & x_1 & + & x_2 & \leqslant 2 \\ & & & & x_1, x_2 & \in \{0,1\}, \end{array} \tag{7.3}$$

where $\{v^1, v^2, v^3, v^4\} = \{(1,1); (1,0); (0,1); (0,0)\}$ is the feasible set and $\{v^1\}$ is the efficient set. The feasible solutions v^2, v^3 and v^4 are subsets of v^1, which implies that for all profit matrices $D \in \mathbb{N}^{q\times n} : Dv^1 \geqq Dv^2$, $Dv^1 \geqq Dv^3$, and $Dv^1 \geqq Dv^4$. Therefore, the inverse problem $[\bullet v^1 \notin E(X,D) \bullet |\cdot|_\infty]$ is not feasible.

Theoretical results

The strategy for solving this problem is to apply the opposite of the approach for solving $[\bullet x^0 \in E(X,D) \bullet |\cdot|_\infty]$. That is to say, the way for building the optimal solution of $[\bullet x^0 \in E(X,D) \bullet |\cdot|_\infty]$ is the opposite of building the optimal solution of $[\bullet x^0 \notin E(X,D) \bullet |\cdot|_\infty]$. The profits that were increased in the first case, will be decreased. Inversely, profits that were decreased in the first case, will be increased. This approach is intuitive, because it is obvious that both inverse problems are the opposite of each other. This section provides the theoretical results that prove this intuition.

Let us first analyze the nature of some optimal solutions of $[\bullet x^0 \notin E(X,D) \bullet |\cdot|_\infty]$. Based on a partition of J defined by $J^0 = \{j \in J : x_j^0 = 0\}$ and $J^1 = \{j \in J : x_j^0 = 1\}$, the first theorem establishes that an optimal solution D^* of $[\bullet x^0 \notin E(X,D) \bullet |\cdot|_\infty]$ can be built by decreasing, or keeping equal, C_{ij}, for all $j \in J^1$ and by increasing, or keeping equal, C_{ij}, for all $j \in J^0$, for all $i \in I$.

Theorem 11. *For every feasible instance of* $[\bullet x^0 \notin E(X,D) \bullet |\cdot|_\infty]$ *with profit matrix C, there exists an optimal solution* $D^* \in \mathbb{N}^{q\times n}$ *of* $[\bullet x^0 \notin E(X,D) \bullet |\cdot|_\infty]$ *such that* $\forall j \in J^1 : D^*_{ij} \leqslant C_{ij}$ *and* $\forall j \in J^0 : D^*_{ij} \geqslant C_{ij}$*, with* $i \in I$.

Proof. Let $D \in \mathbb{N}^{q\times n}$ denote an optimal solution of $[\bullet x^0 \notin E(X,D) \bullet |\cdot|_\infty]$. Define the following sets for all $i \in I : J_i^{0<} = \{j \in J^0 : D_{ij} < C_{ij}\}$, $J_i^{1>} = \{j \in J^1 : D_{ij} > C_{ij}\}$. By definition of D, there is a feasible solution $x \in X$ with $Dx \geq Dx^0$. Consider a matrix $D^* \in \mathbb{N}^{q\times n}$ defined as follows, for all $i \in I, j \in J$:

$$D^*_{ij} := \begin{cases} C_{ij}, & \text{if } j \in \{J_i^{1>} \cup J_i^{0<}\}, \\ D_{ij}, & \text{otherwise.} \end{cases} \tag{7.4}$$

Let us show that if $Dx \geq Dx^0$, then $D^*x \geq D^*x^0$. This is equivalent to show that the following condition holds: if $D(x - x^0) \geq 0$, then $D^*(x - x^0) \geq 0$. The profit matrix D^* is introduced into the first inequality as follows, $(D + D^* - D^*)(x - x^0) \geq 0$, which leads to write: $D^*(x - x^0) \geq (D^* - D)(x - x^0)$.

From Equation 7.4 and the definition of J^1, one may deduce that, for all $j \in J^1$, $(x-x^0)_j \leqslant 0$ and $(D^*_{ij} - D_{ij}) \leqslant 0$. Similarly, for all $j \in J^0$, $(x-x^0)_j \geqslant 0$ and $(D^*_{ij} - D_{ij}) \geqslant 0$. Therefore, $(D^*-D)(x-x^0) \geqq 0$ and consequently $D^*(x-x^0) \geq 0$. This implies that D^* is a feasible solution of $[\bullet x^0 \notin E(X,D) \bullet |\cdot|_\infty]$ and therefore an optimal one, because the inequality $|D^*-C|_\infty \leqslant |D-C|_\infty$ is directly deduced from Equation 7.4. □

Based on the same principle of increasing and decreasing some specific parts of the profit matrices, let us define a particular operator $\ominus$ between pairs of matrices with respect to x^0. In order to not overload the notation, this feasible solution x^0 is omitted .

Definition 28 ($D \ominus E$). *Let D and E be two matrices of size $q \times n$. For all $i \in I$ and $j \in J$,*

$$(D \ominus E)_{ij} := \begin{cases} \max\{0, D_{ij} - E_{ij}\}, & \text{if } j \in J^1, \\ D_{ij} + E_{ij}, & \text{otherwise.} \end{cases}$$

This particular operation is crucial, because when it is applied on a feasible solution of $[\bullet x^0 \notin E(X,D) \bullet |\cdot|_\infty]$, the resulting solution is also feasible. This is established in the following theorem.

Theorem 12. *For all $E \in \mathbb{N}^{q\times n}$, if D is a feasible solution of $[\bullet x^0 \notin E(X,D) \bullet |\cdot|_\infty]$, then $D \ominus E$ is also feasible.*

Proof. Let us show that if $Dx \geq Dx^0$, then $(D \ominus E)x \geq (D \ominus E)x^0$. Similarly to the proof of Theorem 11, it is equivalent to show that the following condition holds: if $D(x-x^0) \geq 0$, then $(D \ominus E)(x-x^0) \geq 0$. Consequently, one can write: $(D \ominus E)(x-x^0) \geq [(D \ominus E) - D](x-x^0) \geq 0$.

From Definition 28, one may deduce that, for all $i \in I, j \in J^1$, $(x-x^0)_j \leqslant 0$ and $[(D \ominus E) - D]_{ij} \leqslant 0$. Similarly, for all $i \in I, j \in J^0$, $(x-x^0)_j \geqslant 0$ and $[(D \ominus E) - D]_{ij} \geqslant 0$. Therefore, $[(D \ominus E) - D](x-x^0) \geqq 0$ and consequently $(D \ominus E)(x-x^0) \geq 0$. This implies the proof of this theorem. □

Let us define a matrix, denoted $D^k \in \mathbb{N}^{q\times n}$, of distance at most k from matrix C with respect to the L_∞ distance.

Definition 29 (D^k). *Let $k \geqslant 0$ be a natural number. Then, D^k is a matrix of size $q \times n$, where for all $i \in I$ and $j \in J$,*

$$D^k_{ij} := \begin{cases} \max\{0, C_{ij} - k\}, & \text{if } j \in J^1, \\ C_{ij} + k, & \text{otherwise.} \end{cases}$$

The following theorem provides an optimality condition for D^k based on the value of k.

Theorem 13. *If there exists an optimal solution D of $[\bullet x^0 \notin E(X,D) \bullet |\cdot|_\infty]$ with $|C-D|_\infty = k$, then D^k is also an optimal solution of $[\bullet x^0 \notin E(X,D) \bullet |\cdot|_\infty]$.*

Proof. From Theorem 11, it can be assumed that $\forall i \in I$, $\forall j \in J^1 : D_{ij} \leqslant C_{ij}$ and $\forall j \in J^0 : D_{ij} \geqslant C_{ij}$. Therefore, it is easy to build a matrix $E \in \mathbb{N}^{q \times n}$ such that $D \ominus E = D^k$. From Theorem 12, D^k is a feasible solution of $[\bullet x^0 \notin E(X,D) \bullet |\cdot|_\infty]$ and obviously an optimal solution to $[\bullet x^0 \notin E(X,D) \bullet |\cdot|_\infty]$. □

As a corollary of this theorem, an optimal solution of $[\bullet x^0 \notin E(X,D) \bullet |\cdot|_\infty]$ can be built on the basis of the optimal solution value. Indeed, if the optimal solution value of $[\bullet x^0 \notin E(X,D) \bullet |\cdot|_\infty]$ is equal to k, then an optimal solution of this problem is given by the matrix D^k. Therefore, $[\bullet x^0 \notin E(X,D) \bullet |\cdot|_\infty]$ can be reduced to finding the optimal solution value, which is given by the minimal value of k where x^0 is a non-efficient solution with respect to D^k. In order to reduce the search domain, an upper bound on this value is provided in the following lemma.

Lemma 8. *If* $\delta_\infty \in \mathbb{N}$ *is the optimal solution value of* $[\bullet x^0 \notin E(X,D) \bullet |\cdot|_\infty]$, *then* $\delta_\infty \leqslant \Delta = \max_{i \in I, j \in J}\{x^0_j C_{ij}\}$

Proof. For a feasible instance of $[\bullet x^0 \notin E(X,D) \bullet |\cdot|_\infty]$, it is always possible to build a matrix $D \in \mathbb{N}^{q \times n}$ such that $|C - D|_\infty = \max_{i \in I, j \in J}\{x^0_j C_{ij}\}$ and Dx^0 is a dominated solution of (X, D). The matrix is defined as follows, $\forall i \in I, \forall j \in J^0 : D_{ij} = C_{ij}$ and $\forall i \in I, \forall j \in J^1 : D_{ij} = 0$. It is easy to see that there exists another feasible solution $x \in X$ such that $Dx \geq Dx^0$ and $|C - D|_\infty = \max_{i \in I, j \in J}\{x^0_j C_{ij}\}$. This solution satisfies the condition that there exists a $j \in J$, such that $x_j = 1$ and $x^0_j = 0$; otherwise the problem is not feasible. This concludes the proof. □

Algorithm

Based on the results presented in the previous section, an algorithm for computing an optimal solution of $[\bullet x^0 \notin E(X,D) \bullet |\cdot|_\infty]$ is devised. On the basis to Theorem 13, an optimal solution of $[\bullet x^0 \notin E(X,D) \bullet |\cdot|_\infty]$ can be built based on the distance between matrices D^* and C. Since this distance is bounded from above by $\Delta = \max_{i \in I, j \in J}\{x^0_j C_{ij}\}$ (see Lemma 8), the algorithm consists of finding the minimal value $k \in \{1, 2, \ldots, \Delta\}$ such that $D^k x^0$ is a dominated vector of the multi-objective instance (X, D^k). This condition is naturally checked by solving ND-Test.

The minimal value of k can be found by performing a binary search on the set $\{1, 2, \ldots, \Delta\}$, because the value of k can be increased without altering the non-efficiency of x^0 with respect to matrix D^k, as it can be directly deduced from Theorem 12. Therefore, Algorithm 9 requires to solve $log_2 \Delta$ times ND-Test. For more details on the procedure, see the pseudo-code of Algorithm 9.

Computational experiments

The purpose of this section is to report the performance of Algorithm 9 (in terms of CPU time) on several sets of randomly generated instances of the

Algorithm 9 Compute an optimal solution D^* of $[\bullet x^0 \notin E(X,D) \bullet | \cdot |_\infty]$.

```
a ← 0;
b ← Δ;
while ($a \neq b$) do
    $k \leftarrow a + \lfloor (b-a)/2 \rfloor$;
    Build matrix $D^k$;
    $x^* \leftarrow$ Solve ND-Test;
    if ($D^k x^0 \neq D^k x^*$) then
        $b \leftarrow k$;
    else
        $a \leftarrow k+1$;
    end if
end while
$D^* \leftarrow D^a$;
```

bi-objective {0,1}-knapsack problem (BKP).

As in the previous chapter, the design of the experiments is inspired by the frameworks used in Martello and Toth (1990) and in Pisinger (1995). For a given number of variables n and *data range* R, a set of instances was randomly generated in the following way. Each instance $s \in \{1, 2, \ldots, S\}$ is generated with the seed number s. The values of C_{1j}, C_{2j} and w_j are uniformly distributed within the range $[1, R]$, and W is computed as the maximum between R and $\lfloor P \sum_{j \in J} w_j \rfloor$, where $P \in [0, 1]$. Groups were composed of $S = 30$ instances. The choice of 30 is based on the *rule of thumb* in statistics to produce good estimates (Coffin and Saltzman 2000). Other types of randomly generated instances, for which the profit matrices are correlated with the weights, were also considered. Since these instances have led to the same kind of results, we will not detail them hereafter.

The performance of the algorithm was measured through the average (Avg.), standard deviation (Std. dev.), minimum (Min.) and maximum (Max.) of the CPU time in seconds. This algorithm has been implemented in the C# programming language and ND-Test was solved by using the CPLEX solver through the C# library. These experiments were performed on a 3.0 GHz dual-core processor with 4GB of RAM.

For each set of instances, Algorithm 9 was run on each efficient solution. Results show that Algorithm 9 is very efficient for large scale instances. For example, with $n = 500$, $R = 1000$ and $P = 0.5$, the average CPU time is 4.91 seconds with a standard deviation of 2.14. Let us point out that the performances of this algorithm are strongly linked to Problem ND-Test. Therefore, the use of another way to test the efficiency of a feasible solution could either increase or decrease significantly the CPU time. For more details about the results of the computational experiments, the reader may consult Table 7.2.

	CPU time (s)							
	R = 100				R = 1000			
n	Avg.	Std. dev.	Min.	Max.	Avg.	Std. dev.	Min.	Max.
10	0.41	0.10	0.30	0.74	0.58	0.09	0.44	0.81
50	0.51	0.10	0.42	0.96	0.77	0.13	0.62	1.30
100	0.68	0.17	0.51	1.14	1.13	0.29	0.80	2.25
500	1.31	0.81	0.28	3.58	4.91	2.14	0.96	8.32

Table 7.2: Impact of varying the number of variables n and data range R on the performance of Algorithm 9 with a group of instances with $P = 0.5$.

7.5 An integer linear programming formulation of $[\bullet x^0 \notin E(X, D)\bullet]$

An integer linear program for solving $[\bullet x^0 \notin E(X, D)\bullet]$ is proposed here. Let us first start with the following formulation (with non-linear constraints).

$$\begin{aligned} D^* \in \arg\min \quad & \delta(C, D) \\ \text{subject to:} \quad & D \in \mathbb{N}^{q\times n} \\ & x \in X \\ & (Dx)_i \geqslant (Dx^0)_i + s_i, \text{ for all } i \in I \\ & \sum_{i\in I} s_i \geqslant 1 \\ & s_i \geqslant 0, \text{ for all } i \in I. \end{aligned} \tag{7.5}$$

where the set of constraints requires that Dx dominates Dx^0.

In order to linearize the set of constraints, the non-linear vector product $(Dx)_i$ is replaced by a sum of n real-valued variables. Let us create nq variables $e_{ij} \in \mathbb{R}$ with $i \in I, j \in J$. Those variables e_{ij} are either equal to zero if $x_j = 0$, or equal to D_{ij} if $x_j = 1$. This is expressed by the following constraints: $e_{ij} \geqslant 0$, $e_{ij} \leqslant D_{ij}$, $e_{ij} \leqslant x_j M$, $e_{ij} \geqslant D_{ij} - (1 - x_j)M$, where M is an upper bound on the value of D^*_{ij} for all $i \in I, j \in J$. This bound can be obtained by considering a profit matrix $D^0 \in \mathbb{N}^{q\times n}$ defined as follows: D^0 is either equal to zero if $x^0_j = 1$, or equal to C_{ij} if $x^0_j = 0$. It is easy to see that $[\bullet x^0 \notin E(X, D)\bullet]$ is feasible if and only if D^0 is a feasible solution to it. Consequently, the value of M is determined with respect to D^0 and the choice of distance function. The resulting linear program is given by Problem 7.6.

$$
\begin{array}{rl}
\min & \delta(C,D) \\
\text{subject to:} & D \in \mathbb{N}^{q\times n} \\
& x \in X \\
& \sum_{j\in J} e_{ij} \geqslant (Dx^0)_i + s_i, \text{ for all } i \in I \\
& \sum_{i\in I} s_i \geqslant 1 \\
& s_i \geqslant 0, \text{ for all } i \in I \\
& \text{for all } i \in I, j \in J : \\
& e_{ij} \leqslant D_{ij} \\
& e_{ij} \leqslant x_j M \\
& e_{ij} \geqslant D_{ij} - (1-x_j)M \\
& e_{ij} \geqslant 0 \\
& e_{ij} \in \mathbb{R}
\end{array}
\tag{7.6}
$$

7.6 A branch-and-bound approach for $[\bullet x^0 \in E(X,D)\bullet]$

The inverse problem $[\bullet x^0 \in E(X,D)\bullet]$ consists of finding a minimal adjustment of the profit matrix such that a given feasible solution $x^0 \in X$ becomes efficient. This problem can be stated as follows:

$$
\begin{array}{rl}
D^* \in \arg\min & \delta(C,D) \\
\text{subject to:} & x^0 \in E(X,D) \\
& D \in \mathbb{N}^{q\times n}.
\end{array}
\qquad ([\bullet x^0 \in E(X,D)\bullet])
$$

A branch-and-bound approach to solve $[\bullet x^0 \in E(X,D)\bullet]$ is introduced in this section. The purpose of this approach is to be able to solve $[\bullet x^0 \in E(X,D)\bullet]$ for any linearizable distance function. In the tree, each node is a relaxation of the inverse problem. The initial relaxation is $\min\{\delta(C,D) : D \in \mathbb{N}^{q\times n}\}$. This relaxation is very poor, because there are no constraints to ensure the efficiency of x^0. For each node (relaxation), it is easy to check if x^0 is an efficient solution by solving Problem ND-Test. Let x^* be an optimal solution of this problem. There are $(q+1)$ possibilities for improving the relaxation, i.e. to ensure that x^0 is not dominated by x^*. Indeed, $Dx^0 \geqq Dx^*$, and $(Dx^0)_i > (Dx^*)_i$, with $i \in I$, are $(q+1)$ sufficient conditions to ensure that Dx^0 is not dominated by Dx^*.

The branch-and-bound tree is built as follows. The root is $P^0 = \min\{\delta(C,D) : D \in \mathbb{N}^{q\times n}\}$. Let D^0 be an optimal solution to P^0. By solving Problem ND-Test, it is easy to check if x^0 is an efficient solution for an instance (D^0, X). Let x^* be an optimal solution of this problem. If x^0 is a non-efficient solution ($Cx^* \neq Cx^0$), then $(q+1)$ new child nodes are created. The first one is

$P^0 \cup \{Dx^0 \geqq Dx^*\}$. That is to say, the linear program P^0 where the constraint $Dx^0 \geqq Dx^*$ is added to it. The $(1+i)$-th node is $P^0 \cup \{(Dx^0)_i > (Dx^*)_i\}$, with $i \in I$. This process is repeated for each child node.

The optimal solution of each leaf leads to a feasible solution and an upper bound on $\delta(D^*, C)$. Furthermore, the optimal solution value of a given node is a lower bound for the other nodes in the sub-tree rooted at this node. Let δ^* be an upper bound on $\delta(D^*, C)$. Therefore, if the optimal solution value of a node is greater than δ^*, then the sub-tree rooted at that node is pruned. The procedure is described in Algorithm 10 which explores the tree search in a best-first manner.

Algorithm 10 Compute an optimal solution D^* of $[\bullet x^0 \in E(X,D)\bullet]$.

```
$UP \leftarrow \{P^0\}$;
$\delta^*$ is an upper bound on $\delta(D^*, C)$;
while $UP \neq \emptyset$ do
    Select a linear program $P$ from $UP$ with respect to the best-first search;
    if P is feasible then
        Let $D^0$ be an optimal solution to $P$;
        $\delta^0 \leftarrow \delta(D^0, C)$;
        if $\delta^0 < \delta^*$ then
            Let $x^*$ be an optimal solution to Problem ND-Test for the instance $(X, D^0)$;
            if $D^0x^* \neq D^0x^0$ then
                $UP \leftarrow UP \cup \big\{P \cup \{Dx^0 \geqq Dx^*\}\big\}$;
                $UP \leftarrow UP \cup \big\{P \cup \{(Dx^0)_i > (Dx^*)_i\} : i \in I\big\}$;
            else
                $\delta^* \leftarrow \delta^0$;
                $D^* \leftarrow D^0$;
            end if
        end if
    end if
end while
```

7.7 A cutting-plane approach for $[\bullet X^0 \subseteq E(X,D)\bullet]$

The inverse problem $[\bullet X^0 \subseteq E(X,D)\bullet]$ consists of finding a minimal adjustment of the profit matrix such that a given set of feasible solutions becomes efficient. This problem can be stated as follows:

$$\begin{array}{rl} \min & \delta(C, D) \\ \text{subject to:} & X^0 \subseteq E(X, D) \\ & D \in \mathbb{N}^{q \times n}. \end{array} \qquad ([\bullet X^0 \subseteq E(X,D)\bullet])$$

A cutting-plane approach to solve $[\bullet X^0 \subseteq E(X,D)\bullet]$ is introduced in this section. It requires to establish a relaxation of this problem and a way to generate cuts. Let us consider each feasible solution $x^0 \in X^0$. Let $S(x^0) = \{x^1, x^2, \ldots, x^k\} \subseteq X$ be a set composed of k feasible solutions, and $D \in \mathbb{N}^{q\times n}$ a profit matrix. For all $x \in S(x^0)$, the outcome vector Dx^0 is not dominated by Dx if and only if, either $Dx^0 \geqq Dx$, or there exists $i \in I$ with $(Dx^0)_i > (Dx)_i$. It is easy to translate those conditions into a set of constraints. Consequently, a relaxation of $[\bullet X^0 \subseteq E(X,D)\bullet]$ can be defined as follows

$$\begin{aligned}
D^* \in \arg\min \quad & \delta(C,D) \\
\text{subject to:} \quad & D \in \mathbb{N}^{q\times n} \\
& \text{for all } i \in I, x^0 \in X^0, x^l \in S(x^0): \\
& (Dx^0)_i \geqslant (Dx^l)_i - \alpha_l(x^0)M \\
& (Dx^0)_i > (Dx^l)_i - \beta_{l,i}(x^0)M \\
& \text{for all } x^0 \in X^0, x^l \in S(x^0): \\
& \alpha_l(x^0) + \textstyle\sum_{i\in I} \beta_{l,i}(x^0) \leqslant q \\
& \alpha_l(x^0), \beta_{l,i}(x^0) \in \{0,1\}.
\end{aligned} \qquad ([\bullet X^0 \subseteq E(\boldsymbol{S},D)\bullet])$$

Problem $[\bullet X^0 \subseteq E(\boldsymbol{S},D)\bullet]$ is a relaxation of $[\bullet X^0 \subseteq E(X,D)\bullet]$ because it requires that each solution $x^0 \in X^0$ should be efficient with respect to a subset of feasible solutions and not to the whole set. The constraint $(Dx^0)_i > (Dx^l)_i - \beta_{l,i}M$ should be replaced by $(Dx^0)_i \geqslant (Dx^l)_i - \beta_{l,i}M + 1$ in order to have a linear program.

The value M is an upper bound on D^*_{ij} for all $i \in I, j \in J$. This bound is obtained by considering a zero profit matrix $D^{*,0} \in \mathbb{R}^{q\times n}$ defined as follows: $D^{*,0}$ is equal to zero for all $i \in I, j \in J$. It is easy to see that $[\bullet X^0 \subseteq E(X,D)\bullet]$ is feasible if and only if $D^{*,0}$ is a feasible solution to it, because the outcome of all feasible solution in $(X, D^{*,0})$ is equal to the null vector. Based on $D^{*,0}$ and the choice of a distance function, it is easy to determine a value for M. For instance, if $\delta(C,D) = |C-D|_\infty$ in the γ field, then $M = \max\{C_{ij} : i \in I, j \in J\}$.

Let D^0 be an optimal solution to $[\bullet X^0 \subseteq E(\boldsymbol{S},D)\bullet]$. The condition $x^0 \in E(X, D^0)$ can be checked by solving Problem ND-Test. A cut with respect to D^0 is generated in Problem $[\bullet X^0 \subseteq E(\boldsymbol{S},D)\bullet]$ by adding x^* to $S(x^0)$.

Sketch of the algorithm. By solving Problem ND-Test, for all $x^0 \in X^0$, it is easy to check if x^0 is an efficient solution to an instance (X,C). Let x^* be an optimal solution of this problem. If x^0 is not an efficient solution ($Cx^* \neq Cx^0$), then add x^* in the set $S(x^0)$. If for all $x^0 \in X^0 : S(x^0) = \emptyset$, then stop. Otherwise, solve Problem $[\bullet X^0 \subseteq E(\boldsymbol{S},D)\bullet]$. Let D^0 be an optimal solution to this problem. If X^0 is a set of efficient solutions with respect to (X, D^0), then stop; otherwise, repeat the process. See Algorithm 11 for a pseudo-code of this procedure.

Theorem 14. *Algorithm 11 converges to an optimal solution of the inverse problem* $[\bullet X^0 \subseteq E(X, D)\bullet]$ *in a finite number of steps.*

Proof. Let us consider the end of any iteration t of the while loop. For all $x^0 \in X^0$ it is ensured that for all $x \in S(x^0)$, $D^t x^0$ is not dominated by $D^t x$, and there exists an x^* such that $D^t x^* \geqq D^t x^0$. If there exists at least one $x^0 \in X^0$ such that $D^t x^* \neq D^t x^0$, with $x^* \in X$, then the variable *Count* is not equal to zero. Therefore, the while loop continues to the next iteration. This implies that $x^* \notin S(x^0)$, otherwise it would contradict the feasibility of D^t to $[\bullet X^0 \subseteq E(\boldsymbol{S}, D)\bullet]$. Therefore, the number of iterations of the while loop is bounded by the cardinality of the feasible set. At the end of the algorithm, the loop condition ensures that $X^0 \subseteq E(X, D^t)$, and $[\bullet X^0 \subseteq E(\boldsymbol{S}, D)\bullet]$ implies that D^t is an optimal solution. □

Algorithm 11 Compute an optimal solution D^* of $[\bullet X^0 \subseteq E(X, D)\bullet]$.

```
$D^0 \leftarrow C$;
Count $\leftarrow 0$;
for all $x^0 \in X^0$ do
    Solve ND-Test for $(X, D^0)$ and let $x^*$ be an optimal solution;
    if $D^0 x^* \neq D^0 x^0$ then
        $S(x^0) \leftarrow x^*$;
        Count $\leftarrow$ Count $+ 1$;
    else
        $S(x^0) \leftarrow \emptyset$;
    end if
end for
$t \leftarrow 0$;
while Count $> 0$ do
    $t \leftarrow t + 1$;
    Let $D^t$ be an optimal solution of $[\bullet X^0 \subseteq E(\boldsymbol{S}, D)\bullet]$ for $(X, D^{t-1})$;
    Count $\leftarrow 0$;
    for all $x^0 \in X^0$ do
        Solve Problem ND-Test for $(X, D^t)$ and let $x^*$ be an optimal solution;
        if $D^t x^* \neq D^t x^0$ then
            $S(x^0) \leftarrow S(x^0) \cup x^*$;
            Count $\leftarrow$ Count $+ 1$;
        end if
    end for
end while
$D^* \leftarrow D^t$;
```

7.8 A cutting-plane approach for $[\bullet\{Dx^0\} = ND(X,D)\bullet]$

The inverse problem $[\bullet\{Dx^0\} = ND(X,D)\bullet]$ consists of finding a minimal adjustment of the profit matrix such that a given efficient solution becomes an ideal one. This problem can be stated as follows:

$$\begin{array}{rll} D^* \in \arg\min & \delta(C,D) & \\ \text{subject to:} & \{Dx^0\} = ND(X,D) & \qquad ([\bullet\{Dx^0\} = ND(X,D)\bullet]) \\ & D \in \mathbb{N}^{q\times n}. & \end{array}$$

Algorithm

A cutting-plane approach for $[\bullet\{Dx^0\} = ND(X,D)\bullet]$ is introduced in this section. It requires to establish a relaxation of this problem and a way to generate cuts. Let $S = \{x^1, x^2, \ldots, x^k\} \subseteq X$ be a set composed of k feasible solutions, and let $D \in \mathbb{R}^{q\times n}$ be a profit matrix. If the outcome vector Dx^0 dominates Dx, then $Dx^0 \geqq Dx$. It is easy to express those conditions into a set of constraints. Consequently, a relaxation of $[\bullet\{Dx^0\} = ND(X,D)\bullet]$ can be defined as follows.

$$\begin{array}{rll} \min & \delta(C,D) & \\ \text{subject to:} & D \in \mathbb{N}^{q\times n} & \\ & \text{for all } i \in I, x^l \in S : & \qquad ([\bullet\{Dx^0\} = ND(\boldsymbol{S},D)\bullet]) \\ & (Dx^0)_i \geqslant (Dx^l)_i & \end{array}$$

Problem $[\bullet\{Dx^0\} = ND(\boldsymbol{S},D)\bullet]$ is a relaxation of $[\bullet\{Dx^0\} = ND(X,D)\bullet]$, because it requires that x^0 is an ideal solution with respect to S, a subset of feasible solutions, and not with respect to the whole set.

Let D^0 be an optimal solution to $[\bullet\{Dx^0\} = ND(\boldsymbol{S},D)\bullet]$ and $x^{*,i}$ an optimal solution to the single objective optimization problem (X, D^0_i), with $i \in I$. The condition $\{D^0x^0\} = ND(X,D^0)$ can be checked by solving the q single objective optimization problems, *i.e.*, $\{D^0x^0\} = ND(X,D^0)$, if and only if, for all $i \in I : (D^0x^0)_i \geqslant (D^0x^{*,i})_i$. A cut with respect to D^0 is generated in Problem $[\bullet\{Dx^0\} = ND(\boldsymbol{S},D)\bullet]$ by adding to S every $x^{*,i}$ such that $(D^0x^0)_i \ngeqslant (D^0x^{*,i})_i$.

Sketch of the algorithm. By solving all Problems (X, C_i), with $i \in I$, it is easy to check if x^0 is an ideal solution to (X,C). Let $x^{*,i}$ be an optimal solution to (X, C_i). If x^0 is an ideal solution ($(Cx^0)_i \geqslant (Cx^{*,i})_i$, for all $i \in I$), then stop. Otherwise, add to S every $x^{*,i}$ such that $(D^0x^0)_i \ngeqslant (D^0x^{*,i})_i$ and solve Problem $[\bullet\{Dx^0\} = ND(\boldsymbol{S},D)\bullet]$. Let D^0 be an optimal solution to this problem. If x^0 is an ideal solution with respect to (X, D^0), then stop; otherwise, repeat the process. See Algorithm 12 for a pseudo-code of this procedure.

Algorithm 12 Compute an optimal solution D^* of $[\bullet\{Dx^0\} = ND(X, D)\bullet]$.

```
$D^0 \leftarrow C$;
$S \leftarrow \emptyset$;
Count $\leftarrow 0$;
for all $i \in I$ do
    Let $x^*$ be an optimal solution to $(X, D^0_i)$;
    if $(D^0 x^*)_i > (D^0 x^0)_i$ then
        $S \leftarrow \{x^*\} \cup S$;
        Count $\leftarrow$ Count $+ 1$;
    end if
end for
$t \leftarrow 0$;
while Count $> 0$ do
    $t \leftarrow t + 1$;
    Let $D^t$ be an optimal solution of $[\bullet\{Dx^0\} = ND(\boldsymbol{S}, D)\bullet]$ for $(X, D^{t-1})$;
    Count $\leftarrow 0$;
    for all $i \in I$ do
        Let $x^*$ be an optimal solution to $(X, D^t_i)$;
        if $(D^t x^*)_i > (D^t x^0)_i$ then
            $S \leftarrow S \cup \{x^*\}$;
            Count $\leftarrow$ Count $+ 1$;
        end if
    end for
end while
$D^* \leftarrow D^t$;
```

Theorem 15. *Algorithm 12 converges to an optimal solution of* $[\bullet\{Dx^0\} = ND(X, D)\bullet]$ *in a finite number of steps.*

Proof. Let us consider the end of any iterations t of the while loop. For all $x \in S$ it is ensured that $D^t x^0$ is not dominated by $D^t x$. If there exists at least an $x^* \in X$ and an $i \in I$ such that $(D^t x^*)_i > (D^t x^0)_i$, then the variable *Count* is not equal to zero. Therefore, the while loop continues to the next iteration. This implies that $x^* \notin S$, otherwise it would contradict the feasibility of D^t to $[\bullet\{Dx^0\} = ND(\boldsymbol{S}, D)\bullet]$. Therefore, the number of iterations of the while loop is bounded by the cardinality of the feasible set. At the end of the algorithm, the loop condition requires that for all $x \in X : D^t x^0 \geqq D^t x$, and $[\bullet\{Dx^0\} = ND(\boldsymbol{S}, D)\bullet]$ ensure that D^t is an optimal solution. □

7.9 A cutting-plane approach for $[\bullet|ND(X,D)| = 1\bullet]$

The inverse problem $[\bullet|ND(X,D)| = 1\bullet]$ consists of finding a minimal adjustment of the profit matrix such that a feasible ideal solution exists. This problem can be stated as follows:

$$\begin{array}{rl} \min & \delta(C,D) \\ \text{subject to:} & |ND(X,D)| = 1 \\ & D \in \mathbb{N}^{q\times n}. \end{array} \qquad ([\bullet|ND(X,D)| = 1\bullet])$$

Algorithm

A cutting-plane approach for $[\bullet|ND(X,D)| = 1\bullet]$ is introduced in this section. As applied in the previous section, it requires to establish a relaxation of this problem and a way to generate cuts. Let $S = \{x^1, x^2, \ldots, x^k\} \subseteq X$ be a set composed of k feasible solutions, let $D \in \mathbb{R}^{q\times n}$ be a profit matrix, and let $x^0 \in X$ be an unknown feasible solution. For all $x^l \in S$, the outcome vector Dx^0 dominates Dx^l if and only if $Dx^0 \geqq Dx^l$. It is easy to express those conditions into a set of constraints. Consequently, a relaxation of $[\bullet|ND(X,D)| = 1\bullet]$ can be defined as follows

$$\begin{array}{rl} \min & \delta(C,D) \\ \text{subject to:} & D \in \mathbb{N}^{q\times n} \\ & x^0 \in X \\ & \text{for all } i \in I, x^l \in S: \\ & (Dx^0)_i \geqslant (Dx^l)_i \end{array} \qquad ([\bullet|ND(\boldsymbol{S},D)| = 1\bullet])$$

Contrary to the relaxation $[\bullet\{Dx^0\} = ND(\boldsymbol{S},D)\bullet]$, the feasible solution x^0 is unknown. Therefore, the vector product $(Dx^0)_i$ is non-linear. However, it can be linearized by replacing it by a sum of n real-valued variables. Let us create nq variables $e_{ij} \in \mathbb{R}_+$ with $i \in I, j \in J$. Those variables e_{ij} are either equal to zero if $x^0_j = 0$, or they are equal to D_{ij} if $x^0_j = 1$. This is expressed by the following four constraints: $e_{ij} \geqslant 0$, $e_{ij} \leqslant D_{ij}$, $e_{ij} \leqslant x_j M$, $e_{ij} \geqslant D_{ij} - (1 - x^0_j)M$, where M is an upper bound on the value of D^*_{ij} for all $i \in I$ and $j \in J$. This bound can be obtained by considering a profit matrix $D^{*,0} \in \mathbb{N}^{q\times n}$ that was defined in Section 7.7. It is easy to see that $[\bullet|ND(X,D)| = 1\bullet]$ is feasible if and only if $D^{*,0}$ is a feasible solution to it, because the outcome of all feasible solution in $(X, D^{*,0})$ is equal to the null vector. Consequently, the value of M is determined with respect to $D^{*,0}$ and the choice of distance function. The resulting linear program is given by Problem 7.7 as follows.

$$
\begin{aligned}
\min \quad & \delta(C, D) \\
\text{subject to:} \quad & D \in \mathbb{N}^{q \times n} \\
& \text{for all } i \in I, x^l \in S: \\
& \textstyle\sum_{j \in J} e_{ij} \geqslant (Dx^l)_i \\
& \text{for all } i \in I, j \in J: \\
& e_{ij} \geqslant 0 \\
& e_{ij} \leqslant D_{ij} \\
& e_{ij} \leqslant x^0_j M \\
& e_{ij} \geqslant D_{ij} - (1 - x^0_j) M
\end{aligned}
\tag{7.7}
$$

This mixed-integer linear program contains $q|S| + 4qn$ constraints, nq real variables, and nq non-negative integer variables.

Sketch of the algorithm. Let D^0 be an optimal solution to $[\bullet|ND(\boldsymbol{S}, D)| = 1\bullet]$, with S initialized to the efficient set $E(X, C)$. If there exists an ideal solution to (X, D^0), then stop. Otherwise, a cut with respect to D^0 is generated in Problem $[\bullet|ND(\boldsymbol{S}, D)| = 1\bullet]$ by adding $E(X, D^0)$ to S, and repeat the process. See Algorithm 13 for a pseudo-code of this procedure.

Algorithm 13 Compute an optimal solution D^* of $[\bullet|ND(X, D)| = 1\bullet]$.

```
S ← E(X, C);
Let $D^0$ be an optimal solution to $[\bullet|ND(\boldsymbol{S}, D)| = 1\bullet]$;
while $|ND(X, D^0)| > 1$ do
    $S \leftarrow E(X, D^0)$;
    Let $D^0$ be an optimal solution to $[\bullet|ND(\boldsymbol{S}, D)| = 1\bullet]$;
end while
$D^* \leftarrow D^0$;
```

7.10 Conclusion

In this chapter, we have extended inverse optimization to multi-objective combinatorial optimization. Different inverse problems have been defined showing the extent of this new field of research. These problems have been solved by branch-and-bound, branch-and-cut, binary search, and integer programming approaches. However, some of these approaches are computationally expensive.

Two illustrative applications are considered in the next chapter. One of the inverse problems considered in this chapter will be used in stability analysis. Two other inverse problems will be used to compute compromise solutions in portfolio selection with multiple experts.

Chapter 8

Illustrative applications

We have identified two fields of application where inverse multi-objective optimization is of a significant interest. In stability analysis, it can be used to assess the ability of a feasible solution to remain efficient when the problem parameters are perturbed. In group decision analysis, it can be used to compute compromise solutions among different experts. In this chapter, we discuss these applications in their practical and theoretical aspects.

8.1 Stability analysis

As already stressed in the introduction, modeling a multi-objective optimization problem requires to fix the values of the parameters in order to define the objective functions and the feasible set. Those values rely on various hypotheses as well as on the accuracy of the evaluations. These sources of uncertainty must be taken into account, because a small perturbation on the model can potentially transform an efficient solution into a non-efficient one. This reflects an instability aspect of the model. A way of assessing such an instability is to compute a stability radius for each efficient solution. This radius is defined as the maximal variation of the problem parameters that allows the solution to remain an efficient one (see Emelichev et al. 2004, Emelichev and Kuzmin 2006, Emelichev and Podkopaev 2010).

The purpose of this section is to study the calculation of the stability radius in the context of multi-objective combinatorial optimization and to investigate how inverse optimization can be used to compute it. In single-objective combinatorial optimization, the stability radius can be computed in polynomial time if the problem is polynomially solvable (Chakravarti and Wagelmans 1998). However, to the best of our knowledge, no algorithm can compute the stability radius for multi-objective combinatorial problems except by explicit enumeration (Emelichev and Podkopaev 2010). Theoretical results are available for measuring the stability radius, but the proposed formula requires the complete enumeration of subsets of the feasible set (Emelichev et al. 2004, Emelichev and Kuzmin 2006, Emelichev and Podkopaev 2010).

Computing the stability radius by inverse optimization

Let us define the stability radius of an efficient solution of a MOCO with respect to the L_∞ distance. The set of all matrices D with distance at most $\epsilon \in \mathbb{N}$ from C is defined by $\Gamma(\epsilon) = \{D \in \mathbb{N}^{q\times n} : |C - D|_\infty \leqslant \epsilon\}$. The *stability radius* of an efficient solution $x \in E(X, C)$ is the optimal solution of $\max\{\epsilon \in \mathbb{N} : \forall D \in \Gamma(\epsilon), x \in E(X, D)\}$.

The calculation of the stability radius is closely related to inverse multi-objective optimization. Indeed, it is easy to see that the stability radius of an efficient solution can be obtained through the minimal adjustment of the parameters, in such a way that a given solution becomes non-efficient. This precise question has been covered in Chapter 7 when we solved $[\bullet x^0 \notin E(X,D) \bullet |\cdot|_\infty]$ by a binary search. The optimal solution value of $[\bullet x^0 \notin E(X,D) \bullet |\cdot|_\infty]$ is equal to the stability radius of x^0 increased by one unit.

The stability radius of an efficient solution can also be measured by any linearizable distance function. In this case, the radius can be computed by solving the inverse problem $[\bullet x^0 \notin E(X,D) \bullet]$. This problem is formulated as an IP in Chapter 7. However, in the next subsection we will only consider the L_∞ norm.

Illustrative Examples

The purpose of this section is to present and analyze the stability radius for several illustrative instances of the bi-objective {0,1}-knapsack problem. An extended version of the stability radius is considered, where we take into account stable and unstable components in the profit matrix, *i.e.*, some values in the profit matrix may have a different value while others are fixed. This helps to get a better understanding of the concept of stability radius. Let us note that we could even consider intervals on the profits to express more precisely the uncertainty on their adequate values, *i.e.*, to consider the inverse problem $[LD_i \leqq D_i \leqq UD_i \text{ for all } i \in I, D \in \mathbb{N}^{q\times n} \bullet x^0 \notin E(X,D) \bullet |\cdot|_\infty]$, where $UD, LD \in \mathbb{N}^{q\times n}$. This would enable a more sophisticated analysis for a real-world application.

The set of stable components in the profit matrix is denoted by $S \subseteq I \times J$. The set of unstable components is denoted by $\bar{S}$, with $\bar{S} \cup S = I \times J$. This leads to modify $[\bullet x^0 \notin E(X,D) \bullet |\cdot|_\infty]$ as follows:

$$\begin{array}{rl} \min & |C - D|_\infty \\ \text{subject to:} & x^0 \notin E(X,D) \\ & D \in \mathbb{N}^{q\times n} \\ & C_{ij} = D_{ij}, \text{ for all } (i,j) \in S. \end{array} \tag{8.1}$$

With the notation $[\alpha \bullet \beta \bullet \gamma]$, this problem is denoted by $[D \in \mathbb{N}^{q\times n}, \forall (i,j) \in S : D_{ij} = C_{ij} \bullet x^0 \notin E(X,D) \bullet |\cdot|_\infty]$. Theorems 11, 12, and 13 can be extended to Problem 8.1. It requires to modify the definition of D^k as follows, for all $i \in I$, $j \in J$,

$$D^k_{ij} := \begin{cases} \max\{0, C_{ij} - k\}, & \text{if } j \in J^1 \text{ and } (i,j) \in \bar{S}, \\ C_{ij} + k, & \text{if } j \in J^0 \text{ and } (i,j) \in \bar{S}, \\ C_{ij}, & \text{otherwise,} \end{cases}$$

as well as the definition of $D \ominus E$, for all $i \in I$, $j \in J$,

$$(D \ominus E)_{ij} := \begin{cases} \max\{0, D_{ij} - E_{ij}\}, & \text{if } j \in J^1 \text{ and } (i,j) \in \bar{S}, \\ D_{ij} + E_{ij}, & \text{if } j \in J^0 \text{ and } (i,j) \in \bar{S}, \\ D_{ij}, & \text{otherwise.} \end{cases}$$

It is easy to show that Theorems 11, 12, and 13 remain valid with these modifications. However, Lemma 8 cannot be applied in this situation as illustrated in the following instance:

$$\begin{array}{rrrrrl} \max & f_1(x) = & 1x_1 & + & 2x_2 & \\ \max & f_2(x) = & 4x_1 & + & 1x_2 & \\ \text{subject to:} & & x_1 & + & x_2 & \leqslant 1 \\ & & & & x_1, x_2 & \in \{0,1\}, \end{array} \tag{8.2}$$

where $\{v^1, v^2, v^3\} = \{(1,0);(0,1);(0,0)\}$ is the feasible set, $\{v^1, v^2\}$ is the efficient set. Lemma 8 states that the optimal solution of $[\bullet v^2 \notin E(X,D) \bullet | \cdot |_\infty]$ is bounded by $\max_{i\in I, j\in J}\{v^2_j C_{ij}\}$ that is equal to 2. However, if $S = \{(1,1);(1,2);(2,2)\}$ is the set of stable components, then there is no feasible solution to Problem 8.1. In this case, by convention, the stability radius of v^2 is equal to infinity. However, the use of Algorithm 9, to solve Problem 8.1, requires to fix the value of an upper bound on the optimal solution value. For this purpose, in practice, one would consider the value of $n \cdot q \cdot \max_{i\in I, j\in J}\{C_{ij}\}$ as being large enough to represent an adequate upper bound on the stability radius, because this modification would completely change the initial profits.

Consider, as a first illustrative example, the following knapsack instance:

$$\begin{array}{rrrrrrrl} \max & f_1(x) = & 10x_1 & + & x_2 & + & 2x_3 & \\ \max & f_2(x) = & 2x_1 & + & 8x_2 & + & 10x_3 & \\ \text{subject to:} & & x_1 & + & x_2 & + & x_3 & \leqslant 1 \\ & & & & & & x_1, x_2, x_3 & \in \{0,1\}, \end{array} \tag{8.3}$$

where $\{v^1, v^2, v^3, v^4\} = \{(1,0,0);(0,1,0);(0,0,1);(0,0,0)\}$ is the feasible set, $\{v^1, v^3\}$ is the efficient set, and all components are unstable. When solving $[\bullet x^0 \notin E(X,D) \bullet | \cdot |_\infty]$ on each efficient solution, the stability radii of v^1 and v^3 are 3 and 0, respectively. It means that v^1 remains efficient even if one increases or decreases by 3 the profit of each item (keeping such profits as non-negative), whereas there exists a profit matrix with distance 1 that leads to transform v^3 into a non-efficient solution. This strong difference does not appear when looking only at the non-dominated set in the objective space given by $\{(10,2);(2,10)\}$. It is due to the fact that all feasible solutions in Problem

8.3 are independent (the intersections between the feasible sets of items are empty) and to the existence of the non-efficient solution v^2 that is very close to v^3 in the objective space. Indeed, the independence implies that the outcome of a solution can be improved or deteriorated without modifying the outcome of another solution. Therefore, the outcome of v^3 can be deteriorated and the outcome of v^2 can be improved without modifying the outcome of v^1. This explains the difference between v^1 and v^2 in terms of stability.

Let us consider the influence of stable components on this instance. If the profit of the second item is stable, then the stability radius of v^3 is increased to 1, because all the modifications to have v^2 dominate v^3 must be applied on the third item.

Consider, as a second example, the following instance:

$$\begin{array}{rrrrrrrl} \max & f_1(x) = & 10x_1 & + & x_2 & + & x_3 & \\ \max & f_2(x) = & 2x_1 & + & 8x_2 & + & 2x_3 & \\ \text{subject to:} & & 2x_1 & + & x_2 & + & x_3 & \leqslant 2 \\ & & & & & x_1, & x_2, x_3 & \in \{0,1\}, \end{array} \tag{8.4}$$

where $\{v^1, v^2, \ldots, v^5\} = \{(1,0,0); (0,1,0); (0,1,1); (0,0,1); (0,0,0)\}$ is the feasible set, $\{v^1, v^3\}$ is the efficient set, and all components are unstable. Even though the image of the feasible set in the objective space is the same for this second instance, the stability radii are different. Their values are both equal to 3. This is because, in this case, v^2 is a subset of v^3, which implies that any modification to the outcome of v^2 leads to a modification to the one of v^3. In other words, for all profit matrices $D \in \mathbb{N}^{q \times n} : Dv^3 \geqq Dv^2$. This explains why v^3 is more stable in this second example.

Consider, as a third example, the following instance:

$$\begin{array}{rrrrrl} \max & f_1(x) = & 2x_1 & + & 4x_2 & \\ \max & f_2(x) = & 4x_1 & + & 2x_2 & \\ \text{subject to:} & & x_1 & + & x_2 & \leqslant 1 \\ & & & & x_1, x_2 & \in \{0,1\}, \end{array} \tag{8.5}$$

where $\{v^1, v^2, v^3\} = \{(1,0); (0,1); (0,0)\}$ is the feasible set, $\{v^1, v^2\}$ is the efficient set, and the set of stable components is $S = \{(1,1); (2,1)\}$ (the first item's profits are stable). The stability radii of v^1 and v^2 are both equal to 1. This shows that even though v^1 is only composed of a single stable item and v^2 of a single unstable item, both have the same stability radius. If all components are unstable, then the stability radii of v^1 and v^2 are both equal to 0. This shows the influence of stable components on the stability radius.

Even though the situations presented in this section can be easily tackled, we should remember that such an analysis would become tedious in large scale instances with complex combinatorial structures. This also shows the usefulness of computing the stability radius for providing information on the underlying structure of the feasible set.

8.2 Compromise solutions in project portfolio selection with multiple experts

Project portfolio selection is a common problem that frequently includes the evaluation of each project by multiple experts (Tian et al. 2005, Shih et al. 2005). It requires to select a subset of projects that satisfies a set of constraints and represents a compromise among the group of experts. Traditional constraints are budget satisfaction and project dependencies.

In this context, let us assume that the expert's evaluations are of a quantitative nature as illustrated in the following example. Consider a company that engages a call for proposals for R&D projects that would be operated in the forthcoming years. Each project is then evaluated by estimating its net present value (NPV). However, this quantity is non-unique, because it requires fixing the rate of return and the period to observe. It might be calculated over a week, a month, etc. As a result, a group of experts is hired to assess the proposals. It is decided that each expert can choose how the net present value is evaluated. Based on these evaluations, the portfolio must maximize the net present value of each expert and satisfy a budget constraint. This decision making task is modeled as a multi-objective combinatorial optimization problem, where each objective function is an expert's evaluation.

A usual approach to tackle this problem consists of building a utility function to quantify the performance of each portfolio. A simple way to build such a utility function would be to aggregate the experts evaluations by the geometric or the arithmetic mean (Forman and Peniwati 1998, Saaty 2005, 1999). In these cases, the evaluations of each expert are aggregated in such a way that the group may be seen as a new "individual". This leads to a classical project portfolio problem where only one decision maker is involved.

Another way to deal with this problem is to model it as a multi-objective combinatorial optimization problem where each objective function is an expert's point of view over the portfolios. This judgment is the sum of the expert's evaluation of the projects selected in the portfolio. A portfolio is said "ideal" if it maximizes the judgments of all experts. Let us observe that if there exists an ideal portfolio, then there is a consensus among the experts on this portfolio. Based on this observation, several concepts of compromise portfolios can be defined. The first one is to find a portfolio (or a set of portfolios) that is as close as possible to the ideal portfolio (Zeleny 1974). This leads to a set of compromises, which depend on the choice of a distance function. Let us consider the efficient set and the ideal solution in Figure 8.1. In this example, the compromise solution with respect to the Euclidean distance is v_3, because it is the closest solution to $\bar{f}$.

A second concept of compromise is presented in this section, which shows how we can use inverse optimization in this context. It is based on the following observation: evaluations are not necessarily precise and a slight modification of their value could be accepted. Hence, the compromise solution may be determined by finding a minimal adjustment of the experts' evaluations so that an

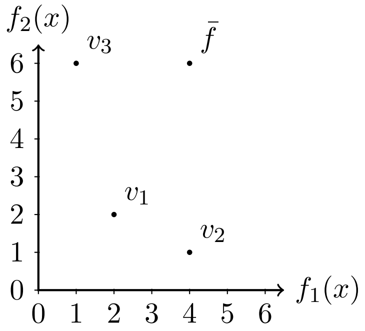

Figure 8.1: A set of efficient solutions $\{v_1, v_2, v_3\}$ and the ideal outcome vector $\bar{f}$

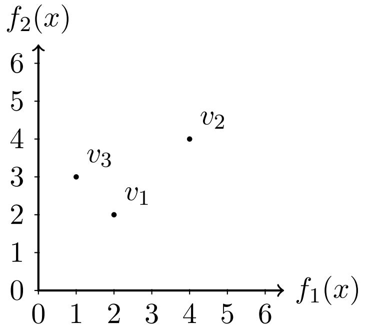

Figure 8.2: A set of feasible solutions $\{v_1, v_3\}$ and the ideal solution v_2

ideal portfolio exists. This leads to a new multi-objective optimization model that is as close as possible to the original one, where there exists an ideal solution. For example, this could lead to transform the problem of Figure 8.1 into the problem of Figure 8.2, where v_2 is the compromise solution. Our approach also allows, in a certain sense, to measure how the experts are conflicting with each other and could indicate potential conflicts among experts on the evaluations of the projects. This information could be used by experts during a negotiation phase.

This concept of compromise is closely related to inverse multi-objective optimization. Indeed, it is easy to see that a compromise can be obtained through the minimal adjustment of the parameters, in such a way that the cardinality of the image of the efficient set in the objective space is equal to one. This precise question has been covered in Chapter 7 when we solved $[\bullet | ND(X, D)| = 1 \bullet]$. A second algorithm can be used to compute the compromise solution. It requires to solve the inverse problem $[\bullet \{Dx^0\} = ND(X, D) \bullet]$ for each feasible solution, where the inverse problem is solved by a cutting plane approach.

These two methods are studied from a theoretical and a practical point of view. Several properties, such as influence of non-discriminating experts, monotonicity, and dominance are proved to be satisfied. Then, the compromise solutions of an illustrative example are analyzed and compared to the ones obtained by the Zeleny's procedure.

Concepts and notation

Let $I = \{1, 2, \ldots, i, \ldots, q\}$ denote a set of experts, and $J = \{1, 2, \ldots, j, \ldots, n\}$ a set of items. The evaluation of an expert $i \in I$ over each item $j \in J$ is denoted by $C_{ij} \in \mathbb{N}$. The portfolio selection problem (PSP) consists of selecting a subset $S \subseteq J$, such that the sum of the evaluations of the elements belonging to S is "maximized" and simultaneously satisfies a set of constraints. The set of constraints is defined by the system $Ax \leqq b$, where $x \in \{0,1\}^n$ is the incidence vector of S, $A \in \mathbb{R}^{m\times n}$, and $b \in \mathbb{R}^m$. The problem can be stated as follows:

$$\begin{aligned} \text{“max”} \quad & F(x) = \{f_1(x), f_2(x), \ldots, f_i(x), \ldots, f_q(x)\} \\ \text{subject to:} \quad & Ax \leqq b \\ & x_j \in \{0,1\},\ j \in J. \end{aligned} \tag{PSP}$$

where $f_i(x) = \sum_{j\in J} C_{ij}x_j$, for all $i \in I$. Consequently, an instance of the PSP problem is a particular multi-objective combinatorial optimization problem. Indeed, it is defined by a feasible set $X \subseteq \big\{x \in \{0,1\}^n\big\}$, and a profit matrix $C \in \mathbb{N}^{q\times n}$. In what follows we will denote (X,C) such an instance.

The relative importance of each expert $i \in I$ is assumed to be given by a weight $\lambda_i \geqslant 0$, with $\sum_{i\in I} \lambda_i = 1$. Hence, two well-known procedures could be applied to handle such a situation. The first one consists of computing the geometric mean of evaluations, for all projects $j \in J$:

$$G(C_j) = \prod_{i\in I} C_{ij}^{\lambda_i}.$$

The second procedure consists of computing the arithmetic mean of the evaluations, for all projects $j \in J$:

$$A(C_j) = \sum_{i\in I} \lambda_i C_{ij}.$$

Those transformations lead to a new PSP reduced to one artificial expert ($q = 1$). The first one leads to the objective function $\sum_{j\in J} G(C_j)x_j$ and the second one leads to $\sum_{j\in J} A(C_j)x_j$.

Another way to handle such a situation would be to find the solutions that are as close as possible to the ideal outcome vector denoted by $\bar{f} = (\bar{f}_1, \bar{f}_1, \ldots, \bar{f}_q)$, where $\bar{f}_i(x) = \max_{x\in X} f_i(x)$. The distance of a feasible solution $x \in X$ to the ideal vector, with respect to an expert $i \in I$, is denoted by $d_i(x) = |f_i(x) - \bar{f}_i|$. Hence, the distance to the ideal vector can be measured as the weighted L_p norm denoted by $L_p(\lambda, x) = [\sum_{i\in I} \lambda_i^p d_i^p]^{1/p}$. Therefore the compromise set is made up of all the feasible solutions that minimize $L_p(\lambda, x)$.

Theoretical developments

The new concept of compromise, based on inverse optimization, is formalized in this section. Let (X, C) denote a PSP and (X, D^*) a minimal adjustment of (X, C) such that there exists an ideal solution in (X, D^*). This ideal solution is a compromise to (X, C). More formally, seeking for a *compromise solution* to a PSP can be stated as follows.

$$\begin{array}{rl} \min & \delta(C, D) \\ \text{subject to:} & |ND(X, D)| = 1 \\ & D \in \mathbb{Z}^{q\times n}. \end{array} \tag{CPSP}$$

where $\delta : \mathbb{R}^{q\times n} \times \mathbb{R}^{q\times n} \to \mathbb{R}_+$ denotes a distance function between two profit matrices. Let D^* denote an optimal solution to CPSP. Every solution $x^* \in X$, such that for all $x \in X : D^*x^* \geqq D^*x$ is a compromise solution to PSP.

As explained previously, this model is closely related to the inverse multi-objective optimization problem $[\bullet\{Dx^0\} = ND(X, D)\bullet]$. Hence a compromise can be reached by solving this inverse problem for all feasible solution $x^0 \in X$.

Let us analyze the nature of some optimal solutions of the inverse problem $[\bullet\{Dx^0\} = ND(X, D)\bullet]$. Based on a partition of J defined by $J^0 = \{j \in J : x^0_j = 0\}$ and $J^1 = \{j \in J : x^0_j = 1\}$, the following theorem establishes that an optimal solution D^* of $[\bullet\{Dx^0\} = ND(X, D)\bullet]$ can be built by decreasing, or keeping equal, C_{ij}, for all $j \in J^0$ and by increasing, or keeping equal, C_{ij}, for all $j \in J^1$, for all $i \in I$.

Theorem 16. *For every instance of* $[\bullet\{Dx^0\} = ND(X, D)\bullet]$ *with profit matrix* C*, there exists an optimal solution* $D^* \in \mathbb{N}^{q\times n}$ *of* $[\bullet\{Dx^0\} = ND(X, D)\bullet]$ *such that* $\forall j \in J^0 : D^*_{ij} \leqslant C_{ij}$ *and* $\forall j \in J^1 : D^*_{ij} \geqslant C_{ij}$*, with* $i \in I$.

Proof. Let $D \in \mathbb{N}^{q\times n}$ denote any optimal solution of $[\bullet\{Dx^0\} = ND(X, D)\bullet]$. Define the following sets for all $i \in I : J^{1<}_i = \{j \in J^1 : D_{ij} < C_{ij}\}$, $J^{0>}_i = \{j \in J^0 : D_{ij} > C_{ij}\}$. By definition of D, $Dx^0 \geqq Dx$, for all $x \in X$. Consider a matrix $D^* \in \mathbb{N}^{q\times n}$ defined as follows, for all $i \in I, j \in J$:

$$D^*_{ij} := \begin{cases} C_{ij}, & \text{if } j \in \{J^{0>}_i \cup J^{1<}_i\} \\ D_{ij}, & \text{otherwise.} \end{cases} \tag{8.6}$$

Let us show that if $Dx^0 \geqq Dx$, then $D^*x^0 \geqq D^*x$. This is equivalent to show that the following condition holds: if $D(x^0 - x) \geqq 0$, then $D^*(x^0 - x) \geqq 0$. The profit matrix D^* is introduced into the first inequality as follows, $(D+D^*-D^*)(x^0-x) \geqq 0$, which leads to write: $D^*(x^0-x) \geqq (D^*-D)(x^0-x)$. From Equation 8.6 and the definition of J^1, one may deduce that, for all $j \in J^1$, $(x^0 - x)_j \geqslant 0$ and $(D^*_{ij} - D_{ij}) \geqslant 0$. Similarly, for all $j \in J^0$, $(x^0 - x)_j \leqslant 0$ and $(D^*_{ij} - D_{ij}) \leqslant 0$. Therefore, $(D^* - D)(x^0 - x) \geqq 0$ and consequently $D^*(x^0 - x) \geqq 0$. This implies that D^* is a feasible solution of $[\bullet\{Dx^0\} = ND(X, D)\bullet]$ and therefore an optimal one y definition of D^* (see Equation 8.6). □

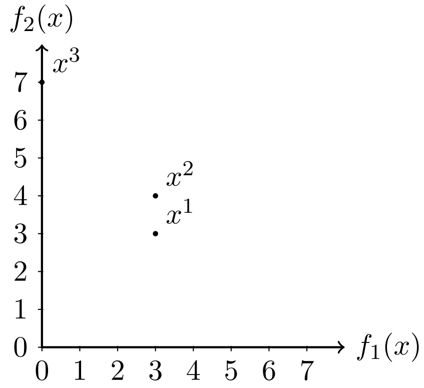

Figure 8.3: The set of feasible solutions of the PSP given in Equation 8.7

Let us illustrate this theorem with the following instance (X, C) of PSP:

$$C = \begin{pmatrix} 1 & 1 & 1 & 3 & 0 \\ 1 & 1 & 1 & 4 & 7 \end{pmatrix}, \qquad X = \begin{cases} x^1 = (1,1,1,0,0) \\ x^2 = (0,0,0,1,0) \\ x^3 = (0,0,0,0,1) \end{cases}, \tag{8.7}$$

where $Cx^1 = (3,3), Cx^2 = (3,4), Cx^3 = (0,7)$, and $E(X,C) = \{x^2, x^3\}$. The image of the efficient set in the objective space is presented in Figure 8.3. If the L_∞ norm is considered, then, from Theorem 16, the best profit matrices at distance 1 from C so that x^1, x^2, or x^3 would become ideal are C^1, C^2, and C^3, respectively.

$$C^1 = \begin{pmatrix} 2 & 2 & 2 & 2 & 0 \\ 2 & 2 & 2 & 3 & 6 \end{pmatrix}, \tag{8.8}$$

$$C^2 = \begin{pmatrix} 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 5 & 6 \end{pmatrix}, \tag{8.9}$$

$$C^3 = \begin{pmatrix} 0 & 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 3 & 8 \end{pmatrix}. \tag{8.10}$$

These profit matrices imply that only x^1 becomes an ideal solution with a profit matrix at distance 1 from C. Therefore, C^1 is an optimal solution to CPSP and x^1 is a compromise solution. It is worth stressing that this compromise is not an efficient solution in the initial problem. Therefore, this method does not satisfy the *dominance* property, which can be stated as follows: "If every expert prefers portfolio A over portfolio B, then portfolio B cannot be selected by the group".

Therefore, two algorithms can be developed to solve CPSP. The first one requires to solve $[\bullet\{Dx^0\} = ND(X,D)\bullet]$ for each feasible solution. This algorithm can be adapted to ensure the dominance property by restricting the domain on the efficient set (*i.e.*, by solving $[\bullet\{Dx^0\} = ND(X,D)\bullet]$ on each efficient solution). The second algorithm solves CPSP by a cutting plane approach (*i.e.*, by solving $[\bullet | ND(X,D)| = 1 \bullet]$). However, this second approach does not ensure the dominance property.

Computational experiments

The purpose of this section is to report the performance of Algorithm 12 and Algorithm 13 (in terms of CPU time) for computing compromise solutions. The design of the experiments is the same as the one presented in Section 7.4. Both algorithms were implemented in the C# programming language and linear programs were solved by using the Cplex solver through the C# library.

Algorithm 12 is applied to each efficient solution. The purpose of these experiments is not to compare the performance of both approaches, because they do not answer the same question (*i.e.* only the first procedure ensure the dominance property), but to determine if these algorithms could be used in practice, *i.e.*, if they can find a compromise solution in a reasonable amount of time.

	CPU time (s)							
	R = 100				R = 1000			
n	Avg.	Std. dev.	Min.	Max.	Avg.	Std. dev.	Min.	Max.
10	0.20	0.18	0.01	0.86	0.34	0.27	0.02	1.41
20	0.34	0.26	0.02	1.44	0.55	0.24	0.11	1.02
30	0.43	0.23	0.11	1.08	0.90	0.47	0.21	1.88
40	0.63	0.30	0.20	1.50	1.41	0.64	0.34	2.95
50	0.98	0.47	0.24	2.65	2.03	0.77	0.73	4.18
60	1.59	1.42	0.41	8.34	3.72	1.68	1.08	8.25

Table 8.1: Impact of varying the number of variables n and data range R on performance of Algorithm 12 with a group of instances with $P = 0.5$.

	CPU time (s)							
	R = 100				R = 1000			
n	Avg.	Std. dev.	Min.	Max.	Avg.	Std. dev.	Min.	Max.
10	0.21	0.13	0.02	0.50	0.24	0.09	0.02	0.42
15	0.72	0.61	0.04	2.40	0.94	0.62	0.03	3.54
20	4.39	5.12	0.04	27.21	3.83	3.30	0.39	13.18
25	23.19	22.91	0.04	86.63	16.99	15.60	5.66	88.65

Table 8.2: Impact of varying the number of variables n and data range R on performance of Algorithm 13 with a group of instances with $P = 0.5$.

Computational results are presented in Tables 8.1 and 8.2. When considering Algorithm 12, the influence of R is significant. Indeed, the average computation time is roughly doubled when R is multiplied by 10. In Algorithm 13, the value of R does not influence the computation time. It is easy to see that Algorithm 12 is much more efficient than Algorithm 13 and could be more easy to use in practice. Algorithm 12 is more efficient, because the domain search is reduced to the efficient set. However, this algorithm can become unusable for large scale instances, because the size of the efficient set can be exponential in the size of the problem instance.

Theoretical study of CPSP

The purpose of this section is to study different properties that should be satisfied by the method (Problem CPSP) proposed in this chapter. The first property is called *non-influence of non-discriminating experts* and it can be stated as follows: "A non-discriminating expert has the same point of view on each portfolio. If a portfolio is a compromise, then it remains a compromise if a non-discriminating expert is taken into account (*i.e.*, is added to the group of experts)". The proof of this theorem requires to extend the notation of $[\bullet\{Dx^0\} = ND(X, D)\bullet]$ to any instance of PSP. For example, for an instance (X_a, C_a) it is denoted by $[\bullet\{D_a x^0\} = ND(X_a, D_a) \bullet \delta(C_a, D_a))]$. This inverse problem consists of finding a profit matrix $D_a \in \mathbb{N}^{q\times n}$ as close as possible to C_a such that $D_a x^0$ is an ideal outcome vector with respect to (X_a, D_a).

Theorem 17. *Problem CPSP is not influenced by any non-discriminating expert.*

Proof. Let (X, C) denote a PSP with C a $q \times n$ profit matrix. Let x^0 be a compromise for CPSP. Let $\tilde{C} \in \mathbb{N}^{(q+1)\times n}$ be an extension of C with a non-discriminating expert, where for all $i \in I$ and $j \in J : \tilde{C}_{ij} = C_{ij}$, and the values $\tilde{C}_{(q+1)j}$ are defined in such a way that for all $x, y \in X : (\tilde{C}_{(q+1)})x = (\tilde{C}_{(q+1)})y$. This means that the $(q + 1)$-th expert has the same point of view on each pair x, y of portfolios.

Let D^0 be an optimal solution of $[\bullet\{Dx^0\} = ND(X, D) \bullet \delta(\tilde{C}, D)]$ and D^1 be an optimal solution of $[\bullet\{Dx^1\} = ND(X, D) \bullet \delta(\tilde{C}, D)]$, where $x^1 \in X$ is any portfolio. Let us prove that $\delta(\tilde{C}, D^0) \leqslant \delta(\tilde{C}, D^1)$, for all $x^1 \in X$. This would implies that x^0 is also a compromise solution for $(X, \tilde{C})$.

It is easy to check that $\tilde{C}_{(q+1)} = D^1_{(q+1)} = D^0_{(q+1)}$, because the performance of each portfolio is already the same on this objective and a modification of these profits would only lead to increase the distance. Indeed, let us assume that $D^0_{(q+1)} \neq \tilde{C}_{(q+1)}$. By definition of D^0, $D^0 x^0 \geqq D^0 x$, for all $x \in X$. One may build another matrix $D^{0'} \in \mathbb{N}^{(q+1)\times n}$ such that $D^{0'}_i = D^0_i$ for all $i \in I$ and $D^{0'}_{(q+1)} = \tilde{C}_{(q+1)}$. It is easy to see that $D^{0'} x^0 \geqq D^{0'} x$, because $(\tilde{C}_{(q+1)})x^0 = (\tilde{C}_{(q+1)})x$. Hence, this matrix is also a feasible solution to $[\bullet\{Dx^0\} = ND(X, D) \bullet \delta(\tilde{C}, D)]$.

Therefore, $\delta(\tilde{C}, D^0) \leqslant \delta(\tilde{C}, D^1)$, otherwise it would contradict the hypothesis that x^0 is a compromise to (X, C). This implies that x^0 remains a compromise. □

Let us notice that an expert who has the same evaluation on each project (*i.e.*, an expert $i \in I$ such that $C_{ij} = C_{ik}$, for all $j, k \in J$) is a discriminating expert, because that would express that a bigger portfolio is preferred to a smaller one. Let us illustrate this case with the following instance (X, C) of PSP:

$$C = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 4 & 7 \end{pmatrix}, \qquad X = \begin{cases} x^1 = (1,1,1,0,0) \\ x^2 = (0,0,0,1,0) \\ x^3 = (0,0,0,0,1) \end{cases}. \tag{8.11}$$

In this example, the first expert has the same evaluation on each project. However, this expert is *not* a non-discriminating expert, because the portfolio x^1 is better than portfolios x^2 and x^3.

The second property is called *monotonicity* and is stated as follows: "If a portfolio is not a compromise solution, then it cannot become a compromise without improving the evaluations of the projects belonging to such a portfolio".

Theorem 18. *Problem CPSP satisfies the monotonicity property.*

Proof. Let x^* be a compromise solution, D^* be an optimal solution to $[\bullet\{Dx^*\} = ND(X,D)\bullet]$. Consider a feasible solution $x^0 \in X$ such that $\delta(D^*,C) < \delta(D^0,C)$, where D^0 is an optimal solution to $[\bullet\{Dx^0\} = ND(X,D)\bullet]$ (*i.e.*, x^0 is not a compromise solution).

Let us build a profit matrix, denoted by $\tilde{C}$, such that the evaluations of the projects belonging to x^0 will not be improved. More formally, $\tilde{C} = C + \tilde{E}$, where $\tilde{E} \in \mathbb{R}^{q\times n}$ satisfies the following conditions: for all $i \in I, j \in J : \tilde{E}_{ij} \leqslant 0$ and for all $j \in J$, if $x^0_j = 0$, then for all $i \in I : \tilde{E}_{ij} = 0$. This transformation of C does not improve the evaluations of the projects belonging to x^0.

Let $\tilde{D}^*$ denotes an optimal solution to $[\bullet\{Dx^*\} = ND(X,D) \bullet \delta(\tilde{C},D)]$ and $\tilde{D}^0$ an optimal solution to $[\bullet\{Dx^0\} = ND(X,D) \bullet \delta(\tilde{C},D)]$. The solution $\tilde{D}^0$ is a transformation of $\tilde{C}$, because it is an optimal solution to $[\bullet\{Dx^0\} = ND(X,D) \bullet \delta(\tilde{C},D)]$. Therefore, $\tilde{D}^0 = \tilde{C} + E^0$, where $E^0 \in \mathbb{N}^{q\times n}$. This leads to write $\tilde{D}^0 = C + \tilde{E} + E^0$, because $\tilde{C} = C + \tilde{E}$. Similarly, one may write $\tilde{D}^* = C + \tilde{E} + E^*$.

Let us assume that thanks to the transformation of C into $\tilde{C}$, x^0 becomes a better potential compromise solution than x^*, *i.e.*, $\delta(\tilde{D}^*,\tilde{C}) > \delta(\tilde{D}^0,\tilde{C})$. Therefore, $\tilde{D}^0x^0 \geq \tilde{D}^0x^*$. If true, this would contradict the monotonicity property. This is equivalent to:

$$(C + \tilde{E} + E^0)x^0 \geq (C + \tilde{E} + E^0)x^*. \tag{8.12}$$

After transformation, it leads to the following equation:

$$(C + E^0)x^0 \geq (C + E^0)x^* + \tilde{E}(x^* - x^0). \tag{8.13}$$

From the definition of $\tilde{E}$, we may deduce that, for all $j \in J$ with $x^0_j = 1$: $(x^* - x^0)_j \leqslant 0$ and $\tilde{E}_{ij} \leqslant 0$. Similarly, for all $j \in J$ with $x^0_j = 0$, $(x^* - x^0)_j \geqslant 0$ and $\tilde{E}_{ij} = 0$. Therefore, $\tilde{E}(x^* - x^0) \geqq 0$ and consequently $(C + E^0)x^0 \geq (C + E^0)x^*$. Moreover, since the distance function is invariant under a translation operation, it can be deduced that $\delta(C + E^0, C) < \delta(C + E^*, C)$, because $\delta(\tilde{D}^0,\tilde{C}) < \delta(\tilde{D}^*,\tilde{C})$. Therefore, x^0 is a better potential compromise solution than x^*. This contradicts the hypothesis on x^0 (*i.e.*, $\delta(D^*,C) < \delta(D^0,C)$). □

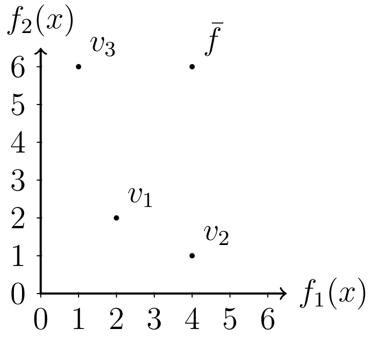

Figure 8.4: Image of the efficient set to Problem 8.14

Illustrative examples

The purpose of this section is to compare the compromise solutions obtained by the methods presented in this paper with the ones obtained by the minimal distance to the ideal. For this purpose, an illustrative example is used to present cases where the methods give different results.

Consider the following knapsack instance:

$$\begin{array}{rrrrrrrl} \max & f_1(x) = & 2x_1 & + & 4x_2 & + & 1x_3 & \\ \max & f_2(x) = & 2x_1 & + & 1x_2 & + & 6x_3 & \\ \text{subject to:} & & x_1 & + & x_2 & + & x_3 & \leqslant 1 \\ & & & & & & x_1, x_2, x_3 & \in \{0,1\}, \end{array} \tag{8.14}$$

where $\{v^1, v^2, v^3\} = \{(1,0,0); (0,1,0); (0,0,1)\}$ is the efficient set. Let us compute the distance between the ideal vector and the image of each efficient solution (see Table 8.3).

Feasible solution	l_1	l_∞
v^1	6	4
v^2	5	5
v^3	3	3

Table 8.3: Distance between each efficient solution and the ideal outcome vector for Problem 8.14.

In this case, the efficient solution v^3 is considered as a compromise for both distance functions. The minimal adjustments to transform these solutions into ideal ones are given in Table 8.4.

With respect to all methods and all measures, v^3 is always preferred to v^2. It is natural, because in both cases the portfolio is evaluated to 1 by an expert and the second evaluation is always better for v^3. When taking into account the distance to the ideal vector, v^3 is always the best solution. However, v^1 could be considered as a compromise too, because it is a balanced solution. This is indeed obtained in the minimal adjustment.

Feasible solution	l_1	l_∞	$\|\|\cdot\|\|_1$	$\|\|\cdot\|\|_\infty$
v^1	6	2	2	4
v^2	5	3	3	5
v^3	3	2	2	3

Table 8.4: Minimal adjustments to transform each efficient solution into an ideal solution for Problem 8.14.

8.3 Conclusion

In this chapter, inverse multi-objective optimization has shown its usefulness in two fields of applications.

In the first application, we have addressed the problem of computing the stability radius of an efficient solution in the context of MOCO. More precisely, we focused on the maximal perturbation of the objective functions coefficients such that a given solution remains efficient. This is measured under the Chebyshev norm and the coefficients are restricted to natural numbers. The stability radius of an efficient solution is modeled as a particular inverse optimization problem. The model is solved by an algorithm which only requires to solve a logarithmic number of mixed integer programs. It contains a linear number of constraints and variables compared with the instance of the combinatorial optimization problem if its feasible set can be defined by linear constraints. To the best of our knowledge, this algorithm is the first one that allows to compute the stability radius in a reasonable amount of time.

In the second application, we have addressed the problem of computing a compromise solution in the context of portfolio selection. More precisely, we focused on the minimal adjustment of the objective functions coefficients of a MOCO in such a way that an ideal solution exists in a multi-objective combinatorial optimization problem. To the best of our knowledge, this is an innovative definition of the concept of compromise.

Two inverse problems are considered to compute such a compromise solution. The first one consists of solving an inverse optimization problem for each efficient solution. The second one is based on a cutting plane algorithm which adds constraints on the profit matrix at each iteration. Even though, the first algorithm is more efficient than the second one, both approaches remain significant because the first one solves CPSP and ensures the dominance property.

We also ensured the good behavior of this approach by a theoretical analysis. We have proved that *monotonicity*, *dominance*, and *influence by non-discriminating expert* properties are satisfied by this approach.

Chapter 9

Conclusion

Before the beginning of this thesis, there was no research studying inverse optimization in the context of multi-objective problems. Motivated by our strong interest in combinatorial optimization, we focused ourselves on inverse combinatorial problems. Because of its simple structure, the knapsack problem was a good starting point for pursuing such a study. However, it was astonishing to discover that the only work available on the inverse knapsack problem was incorrect, yet it was never presented as such in other articles. This is why our first research focused on the inverse knapsack problem.

The inverse knapsack problem has been defined and studied for both the Chebyshev and Manhattan norms. In the case of the Chebyshev norm, the problem has been proved to be **co-NP**-Complete. A pseudo-polynomial time algorithm has been proposed to solve it, which runs in $O(nW \log C)$, where W is the capacity and C is bounded by the largest profit value. The experimental results have shown the efficiency of this procedure for large scale instances. The link between inverse problems and bi-level programming has also been considered. However, the existing approach for solving such bi-level problems cannot be used in practice for large scale instances. In the case of the Manhattan norm, the problem has been proved to be **co-NP**-Hard. This problem has not been proved to belong to **co-NP**, because the building of a certificate requires to solve the knapsack problem. Our approach to solve this problem consisted of formulating it as a linear integer program. However, computational experiments have shown that only small instances can be handled by this approach. The link between inverse problems and bi-level programming has also been considered, but with the same drawbacks in terms of computation time.

With the knowledge brought by this study of the inverse knapsack problem, we were able to face with the challenge of extending inverse optimization to the context of multi-objective problems. We have proposed to define the inverse of a multi-objective optimization instance as the problem of finding a minimal adjustment of this instance inducing a change in the efficient set and/or in the non-dominated set. This broad definition raised many inverse problems that have been described by a triplet. The first component of this triplet describes how the instance is allowed to be modified, the second one provides a set of conditions on the efficient set (and/or on the non-dominated set) of the modified instance, and the third one describes how the modification is measured. This way of describing each inverse problem has shown the extent of this new field of research.

A selection of inverse multi-objective combinatorial optimization problems have been solved. Several proposed approaches have strongly benefited from our research on the inverse knapsack problem. We have based the procedures on traditional optimization approaches such as branch-and-bound, branch-and-cut, binary search, and integer programming. Many inverse problems have been successfully solved. However, some of the proposed approaches were computationally very expensive.

We have identified two fields of application where inverse multi-objective optimization is of a significant interest. This provides a strong motivation for studying inverse multi-objective problems.

In the first application, we have addressed the problem of computing the stability radius of an efficient solution. More precisely, we have focused on the maximal perturbation of the objective functions' coefficients such that a given solution remains efficient. This is computed by solving one of the inverse problems that we had selected. To the best of our knowledge, this approach is the first one that allows to compute the stability radius in a reasonable amount of time.

In the second application, we have addressed the problem of computing a compromise solution in the context of portfolio selection. More precisely, we focused on the minimal adjustment of the objective functions' coefficients in such a way that an ideal solution exists in a multi-objective combinatorial optimization problem. To the best of our knowledge, this is an innovative definition of the concept of compromise. Two approaches based on inverse problems, that we had selected previously, are proposed to address this question. We also ensured the good behavior of this approach by a theoretical analysis. We have proved that *monotonicity*, *dominance*, and *influence by non-discriminating expert* properties are satisfied by this approach.

Perspectives

There are many obvious directions of research that could be pursued. For instance, one may consider the use of other norms, the nature of the adjustment, and constraints on the adjustment.

One of these natural directions of research could be used to address the following project selection problem (started from personal communications with Jeffrey Keisler and Ralph Keeney in 2011, and Roy (1996)). Consider an organization that faces to a project selection problem in R&D with an annual budgeting cycle. Every year a budget is negotiated and new projects are proposed. Then, the organization has to decide on the continuation of projects and the implementation of new ones. Experts in portfolio analysis share the opinion that profits are hard to evaluate and therefore a feasible space for these values rather than a precise one can be provided. The most adequate value should therefore belong to this interval but is unknown till completeness of the project. However, in general, the more a project gets close to its end, the more accuracy can be provided. Therefore profits should be updated each year. As a consequence, the ongoing project portfolio might no longer be an optimal solution. Another important aspect, that should be taken into account, is the interdependencies between projects, because positive synergies or complementarity may occur. Therefore, the organization should take into account the ongoing project portfolio when selecting new ones.

The inverse optimization problem that could help to address this project portfolio selection can be stated as follows. Consider a set of projects and a set of ongoing projects, where the profit of each project is characterized by an interval instead of a precise value. The inverse problem consists of finding a value of profit for each project, that belongs to its respective interval, such that there exists an optimal portfolio which contains the ongoing projects.

The fact that projects are characterized by intervals, also raises the question of preprocessing. For example, one may ask the following question: "Does there exist a profit vector, that is consistent with the intervals, such that a given project belongs to the optimal solution?". This could help to discard some uninteresting projects in order to reduce the search space.

The distinction between supported and unsupported non-dominated solutions should be also investigated. We could try to improve the efficiency of several inverse problems by focusing only on the supported solutions, instead of focusing on all the non-dominated solutions. For example, one may distinguish the computation of non-dominated supported and unsupported solutions.

One could also consider the link between approximation algorithms and inverse optimization in the context of multi-objective problems. Indeed, a new approximation scheme, proposed by Orlin et al. (2008), takes its origin in inverse optimization. It relies on the following principles. Firstly, the parameters of real-world problems are approximate. Secondly, if there exist a small adjustment of these parameters such that a given solution becomes an optimal one, then this solution might be considered to be sufficiently good. As stated previously, this is at the core of inverse optimization. This leads to a new way of defining near-optimal solutions. An approximation algorithm could be designed based on this principle. Such an algorithm could be more efficient than an exact algorithm, because a small change of the parameters can often simplify the problem. It would be worthwhile to study this new scheme in the context of multi-objective optimization and to investigate the potential benefits over traditional approaches.

In this thesis, we have proposed a way to define inverse multi-objective optimization. However, one may consider other ways to define inverse optimization in a multi-objective context. For example, if the adjustment of the optimization instance is performed on two aspects, such as the profit vector and the feasible set, then this adjustment could be evaluated by two distinct objective functions. Our definition of inverse multi-objective optimization could also be extended in a more multi-objective perspective. Indeed, all inverse multi-objective optimization problems that have been considered in this thesis are single-objective optimization problems. An inverse multi-objective optimization problem could be a multi-objective optimization problem as well. For example, the adjustment of each objective function's coefficients could be quantified by a distinct objective function in the inverse problem.

Appendix A

Computational Results

In this appendix are presented the results from the computational experiments for the inverse {0,1}-knapsack problem (Chapter 6). The performance results of Algorithm 7 are presented in Table A.1 and Table A.2. The performance results of Problem 6.3 are presented in Tables A.3, A.4, A.5, and A.6.

	CPU time (s)											
	R = 1000				R = 5000				R = 10000			
n	Avg.	Std. dev.	Min.	Max.	Avg.	Std. dev.	Min.	Max.	Avg.	Std. dev.	Min.	Max.
100	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.01
500	0.00	0.00	0.00	0.01	0.02	0.02	0.00	0.05	0.06	0.05	0.00	0.19
1000	0.01	0.01	0.00	0.03	0.06	0.05	0.01	0.17	0.11	0.10	0.01	0.32
5000	0.09	0.05	0.01	0.22	0.40	0.28	0.01	1.09	0.78	0.66	0.23	2.35
10000	0.23	0.16	0.02	0.51	1.07	0.74	0.31	2.72	2.35	1.77	0.50	6.29
50000	1.19	0.87	0.12	3.08	6.09	5.43	1.87	18.03	15.57	11.67	4.55	41.60
100000	2.76	2.15	0.25	7.22	12.65	11.97	2.20	41.87	27.26	25.13	6.42	88.59

Table A.1: Impact of varying the number of variables n and data range R on performance of Algorithm 7 with a group of strongly correlated instances of Type 2 and $P = 0.5$.

	CPU time (s)											
	R = 1000				R = 5000				R = 10000			
P	Avg.	Std. dev.	Min.	Max.	Avg.	Std. dev.	Min.	Max.	Avg.	Std. dev.	Min.	Max.
0.1	2.42	0.62	0.29	3.10	12.97	4.09	0.26	18.07	30.97	5.62	21.25	39.39
0.2	2.62	0.84	0.24	3.94	16.42	3.30	9.79	21.51	34.69	6.87	25.61	49.98
0.3	2.14	0.94	0.27	4.34	14.46	4.81	0.30	24.41	31.85	9.33	21.57	58.51
0.4	2.25	0.89	1.29	4.63	13.42	6.47	6.44	29.80	33.68	20.86	12.10	78.98
0.5	2.76	2.15	0.25	7.22	12.65	11.97	2.20	41.87	27.26	25.13	6.42	88.59
0.6	2.57	2.26	0.58	8.75	17.01	15.35	2.18	50.21	27.58	21.03	4.36	66.60
0.7	2.27	2.22	0.20	8.07	16.81	16.81	0.96	52.87	37.81	38.97	1.99	161.34
0.8	3.79	3.14	0.27	9.24	19.71	19.07	0.42	68.02	48.22	45.81	0.90	143.98
0.9	3.06	2.82	0.24	8.51	13.82	17.06	0.30	56.60	37.76	37.22	0.46	143.49

Table A.2: Impact of varying the percentage P and data range R on performance of Algorithm 7 with a group of strongly correlated instances of Type 2 and $n = 10^5$.

	CPU time (s)											
	R = 100				R = 300				R = 500			
P	Avg.	Std. dev.	Min.	Max.	Avg.	Std. dev.	Min.	Max.	Avg.	Std. dev.	Min.	Max.
10	0.02	0.00	0.01	0.02	0.05	0.01	0.03	0.08	0.10	0.02	0.05	0.13
20	0.21	0.13	0.10	0.78	0.96	0.80	0.39	3.76	1.78	1.12	0.75	4.74
30	1.07	0.72	0.38	3.64	5.88	5.58	2.06	29.20	9.79	5.99	3.15	27.27
40	2.94	1.39	1.06	6.84	14.27	13.55	5.69	58.55	26.89	26.41	10.22	118.08
50	6.45	6.75	2.37	39.20	36.07	48.34	11.06	250.02	79.41	76.23	19.25	323.14
60	14.38	15.53	4.13	69.71	51.72	52.94	21.90	293.89	186.76	272.88	36.90	1227.79
70	25.59	28.73	7.28	138.20	127.34	151.93	29.34	730.71	336.56	382.42	63.45	1365.06
80	27.78	30.68	11.67	147.03	234.08	300.29	51.13	1091.19	744.64	841.09	110.69	3006.52

Table A.3: Impact of varying the number of variables n and data range R on performance of Problem 6.3 with a group of uncorrelated instances of Type 2 and $P = 0.50$.

	CPU time (s)											
	R = 100				R = 300				R = 500			
P	Avg.	Std. dev.	Min.	Max.	Avg.	Std. dev.	Min.	Max.	Avg.	Std. dev.	Min.	Max.
0.1	0.00	0.00	0.00	0.01	0.01	0.00	0.01	0.02	0.02	0.00	0.02	0.03
0.2	0.00	0.00	0.00	0.01	0.01	0.00	0.01	0.02	0.02	0.00	0.02	0.03
0.3	0.01	0.00	0.00	0.01	0.02	0.00	0.02	0.03	0.04	0.01	0.03	0.06
0.4	0.01	0.00	0.00	0.02	0.04	0.01	0.03	0.06	0.07	0.01	0.03	0.09
0.5	0.02	0.00	0.01	0.02	0.05	0.01	0.03	0.08	0.10	0.02	0.05	0.13
0.6	0.02	0.01	0.01	0.03	0.07	0.02	0.05	0.10	0.12	0.02	0.07	0.16
0.7	0.03	0.01	0.01	0.05	0.08	0.02	0.05	0.13	0.16	0.03	0.08	0.23
0.8	0.03	0.01	0.02	0.05	0.10	0.02	0.06	0.14	0.20	0.04	0.11	0.30
0.9	0.03	0.01	0.02	0.06	0.12	0.03	0.08	0.18	0.23	0.04	0.11	0.31

Table A.4: Impact of varying the percentage P and data range R on performance of Problem 6.3 with a group of uncorrelated instances of Type 2 and $n = 10$.

	CPU time (s)											
	R = 100				R = 300				R = 500			
P	Avg.	Std. dev.	Min.	Max.	Avg.	Std. dev.	Min.	Max.	Avg.	Std. dev.	Min.	Max.
10	0.02	0.01	0.01	0.02	0.05	0.01	0.03	0.08	0.10	0.02	0.06	0.15
20	0.67	0.29	0.15	1.41	2.99	1.70	0.47	7.18	10.06	8.74	2.13	34.32
30	5.50	3.52	0.52	17.78	103.33	76.33	28.33	396.70	271.33	321.63	8.15	1741.39
40	41.82	27.17	3.10	102.82	-	-	-	-	-	-	-	-
50	209.38	158.78	3.54	623.94	-	-	-	-	-	-	-	-

Table A.5: Impact of varying the number of variables n and data range R on performance of Problem 6.3 with a group of strongly correlated instances of Type 2 and $P = 0.50$.

	CPU time (s)											
	R = 100				R = 300				R = 500			
P	Avg.	Std. dev.	Min.	Max.	Avg.	Std. dev.	Min.	Max.	Avg.	Std. dev.	Min.	Max.
0.1	0.00	0.00	0.00	0.01	0.01	0.00	0.01	0.02	0.02	0.01	0.01	0.03
0.2	0.00	0.00	0.00	0.01	0.01	0.00	0.01	0.02	0.02	0.01	0.02	0.03
0.3	0.01	0.00	0.00	0.01	0.02	0.01	0.01	0.04	0.05	0.01	0.03	0.07
0.4	0.01	0.00	0.01	0.02	0.04	0.01	0.02	0.06	0.07	0.02	0.04	0.10
0.5	0.02	0.01	0.01	0.02	0.05	0.01	0.03	0.08	0.10	0.02	0.06	0.15
0.6	0.02	0.01	0.01	0.03	0.07	0.02	0.04	0.10	0.13	0.03	0.08	0.19
0.7	0.03	0.01	0.02	0.04	0.08	0.02	0.05	0.12	0.16	0.04	0.10	0.25
0.8	0.03	0.01	0.02	0.04	0.10	0.02	0.06	0.15	0.20	0.05	0.11	0.32
0.9	0.03	0.01	0.02	0.05	0.11	0.03	0.07	0.17	0.23	0.06	0.12	0.39

Table A.6: Impact of varying the percentage P and data range R on performance of Problem 6.3 with a group of strongly correlated instances of Type 2 and $n = 10$.

Bibliography

Ahuja, R. K., J. B. Orlin. 2001. Inverse optimization. *Operations Research* **49**(5) 771–783.

Ahuja, R. K., J. B. Orlin. 2002. Combinatorial algorithms for inverse network flow problems. *Networks* **40**(4) 181–187.

Bhaskar, K. A. 1979. Multiple objective approach to capital budgeting. *Accounting and Business Research* **10**(37) 25–46.

Burton, D., Ph. L. Toint. 1992. On an instance of the inverse shortest paths problem. *Mathematical Programming* **53**(1) 45–61.

Chakravarti, N., A. P. M. Wagelmans. 1998. Calculation of stability radius for combinatorial optimization. *Operations Research Letters* **23**(1) 1–7.

Coffin, M., M. J. Saltzman. 2000. Statistical analysis of computational tests of algorithms and heuristics. *INFORMS Journal on Computing* **12**(1) 24–44.

Cormen, T. H., C. E. Leiserson, R. L. Rivest, C. Stein. 2001. *Introduction to algorithms*. MIT press.

Dantzig, G., R. Fulkerson, S Johnson. 1954. Solution of a large-scale traveling-salesman problem. *Journal of the Operations Research Society of America* **2**(4) 393–410.

DeNegre, S. T., T. K. Ralphs. 2009. A branch-and-cut algorithm for integer bilevel linear programs. *Operations Research and Cyber-Infrastructure*, *Operations Research/Computer Science Interfaces Series*, vol. 47. Springer, US, 65–78.

Deza, M. M., E. Deza. 2009. *Encyclopedia of Distances*. Springer, Berlin, Germany.

Ehrgott, M. 2005. *Multicriteria optimization*. 2nd ed. Springer, Berlin.

Emelichev, V. A., K. G. Kuzmin. 2006. Stability radius of an efficient solution of a vector problem of integer linear programming in the gölder metric. *Cybernetics and Systems Analysis* **42**(4) 609–614.

Emelichev, V. A., K. G. Kuz'min, A. M. Leonovich. 2004. Stability in the combinatorial vector optimization problems. *Automation and Remote Control* **65**(2) 227–240.

Emelichev, V. A., D. Podkopaev. 2010. Quantitative stability analysis for vector problems of 0-1 programming. *Discrete Optimization* **7**(1-2) 48–63.

Erlebach, T., H. Kellerer, U. Pferschy. 2002. Approximating multiobjective knapsack problems. *Management Science* **48**(12) 1603–1612.

Forman, E., K. Peniwati. 1998. Aggregating individual judgments and priorities with the analytic hierarchy process. *European Journal of Operational Research* **108**(1) 165–169.

Garey, M. R., D. S. Johnson. 1979. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA.

Geoffrion, A. M. 1968. Proper efficiency and the theory of vector maximization. *Journal of mathematical analysis and applications* **22**(3).

Gomory, R. E. 1958. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society* **64**(5) 275–278.

Hadamard, J. 1902. Sur les problèmes aux dérivées partielles et leur signification physique. *Princeton University Bulletin* **13**(49-52) 28.

Heuberger, C. 2004. Inverse combinatorial optimization: A survey on problems, methods, and results. *Journal of Combinatorial Optimization* **8**(3) 329–361.

Horn, R. A., C. R. Johnson. 1990. *Matrix Analysis*. Cambridge University Press.

Huang, S. 2005. Inverse problems of some NP-complete problems. *Algorithmic Applications in Management*, *Lecture Notes in Computer Science*, vol. 3521. Springer Berlin, 422–426.

Jenkins, L. 2002. A bicriteria knapsack program for planning remediation of contaminated lightstation sites. *European Journal of Operational Research* **140**(2) 427–433.

Karmarkar, N. 1984. A new polynomial-time algorithm for linear programming. *Combinatorica* **4**(4) 373–395.

Keller, J. B. 1976. Inverse problems. *The American Mathematical Monthly* **83**(2) 107–118.

Kellerer, H., U. Pferschy, D. Pisinger. 1994. *Knapsack Problems*. Springer-Verlag, Berlin, Heidelberg.

Klingman, D., A. Napier, J. Stutz. 1974. Netgen: a program for generating large scale assignment, transportation, and minimum cost flow problems. *Management Science* **20**(5) 814–821.

Kostreva, M. M., W. Ogryczak, D. W. Tonkyn. 1999. Relocation problems arising in conservation biology. *Computers & Mathematics with Applications* **37**(4-5) 135–150.

Martello, S., P. Toth. 1990. *Knapsack Problems: Algorithms and Computer Implementations*. Wiley, Chichester, UK.

Mathews, G. B. 1897. On the partition of numbers. *Proceedings of the London Mathematical Society*. 28, 486–490.

Mitchell, J. E. 2009. Integer programming: Branch and cut algorithms. C. A. Floudas, P. M. Pardalos, eds., *Encyclopedia of Optimization*. Springer, 1643–1650.

Moore, J. T., J. F. Bard. 1990. The mixed integer linear programming problem. *Operations Research* **38**(5) 911–921.

Nemhauser, G. L., L. A. Wolsey. 1988. *Integer and combinatorial optimization*. Wiley-Interscience, New York, NY, USA.

Orlin, J. B., A. S. Schulz, S. Sengupta. 2008. e-optimization schemes and l-bit precision: Alternative perspectives for solving combinatorial optimization problems. *Discrete Optimization* **5**(2) 550–561.

Padberg, M. 1995. *Linear Optimization and Extensions*. 2nd ed. Springer, Berlin.

Papadimitriou, C. H., K. Steiglitz. 1982. *Combinatorial optimization: algorithms and complexity*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Pisinger, D. 1995. Algorithms for knapsack problems. Ph.D. thesis, Department of Computer Science, University of Copenhagen.

Pisinger, D. 1997. A minimal algorithm for the 0-1 knapsack problem. *Operations Research* **45**(5) 758–767.

Roland, J., J. R. Figueira, Y. De Smet. 2013a. The inverse {0, 1}-knapsack problem: Theory, algorithms and computational experiments. *Discrete Optimization* **10**(2) 181–192.

Roland, J., Y. De Smet, J. R. Figueira. 2012. On the calculation of stability radius for multi-objective combinatorial optimization problems by inverse optimization. *4OR* **10**(4) 379–389.

Roland, J., Y. De Smet, J. R. Figueira. 2013b. Inverse multi-objective combinatorial optimization. *Discrete Applied Mathematics* `http://dx.doi.org/10.1016/j.dam.2013.04.024`.

Rosenblatt, M. J., Z. Sinuany-Stern. 1989. Generating the discrete efficient frontier to the capital budgeting problem. *Operations Research* **37**(3) 384–394.

Roy, B. 1996. *Multicriteria Methodology for Decision Aiding*. Kluwer Academic Publishers, Dordrecht, The Netherlands.

Saaty, T. L. 1999. *Decision Making for Leaders: The Analytic Hierarchy Process for Decisions in a Complex World*. RWS Publications, Pittsburgh, Pennsylvania.

Saaty, T. L. 2005. The analytic hierarchy and analytic network processes for the measurement of intangible criteria and for decision-making. J. Figueira, S. Greco, M. Ehrgott, eds., *Multiple Criteria Decision Analysis: State of the Art Surveys*. Springer Verlag, Boston, Dordrecht, London, 345–408.

Schaefer, A. J. 2009. Inverse integer programming. *Optimization Letters* **3**(4) 483–489.

Schrijver, A. 1986. *Theory of linear and integer programming*. John Wiley & Sons, Inc., New York, NY, USA.

Schrijver, A. 2003. *Combinatorial Optimization — Polyhedra and Efficiency, Volume A: Paths, Flows, Matchings*, *Algorithms and Combinatorics*, vol. 24. Springer-Verlag, Berlin, Heidelberg, New York, Hong Kong, London, Milan, Paris, Tokyo.

Shih, H., L. Huang, H. Shyur. 2005. Recruitment and selection processes through an effective gdss. *Computer & Mathematics with Applications* **50**(10-12) 1543–1558.

Sierksma, G. 2001. *Linear and Integer Programming: Theory and Practice*. 2nd ed. CRC Press, New York, USA.

Sokkalingam, P. T., R. K. Ahuja, J. B. Orlin. 1999. Solving inverse spanning tree problems through network flow techniques. *Operations Research* **47**(2) 291–298.

Steuer, R. E. 1986. *Multiple Criteria Optimization: Theory, Computation and Application*. John Wiley, New York, 546 pp.

Tarantola, A. 1987. *Inverse Problem Theory: Methods for Data Fitting and Model Parameter Estimation*. Elsevier Science Publishers, Amsterdam, The Netherlands.

Teng, J. Y., G. H. Tzeng. 1996. A multiobjective programming approach for selecting non-independent transportation investment alternatives. *Transportation Research Part B: Methodological* **30**(4) 291–307.

Tian, Q., J. Ma, J. Liang, R. C. W. Kwok, O. Liu. 2005. An organizational decision support system for effective r&d project selection. *Decision Support Systems* **39**(3) 403–413.

Vanderbei, R. J. 2008. *Linear Programming: Foundations and Extensions*. 3rd ed. Springer.

Wang, L. 2009. Cutting plane algorithms for the inverse mixed integer linear programming problem. *Operations Research Letters* **37**(2) 114–116.

Wendell, R. E., D. N. Lee. 1977. Efficiency in multiple objective optimization problems. *Mathematical Programming* **12**(1) 406–414.

Wolsey, L. A. 1998. *Integer Programming*. Wiley-Interscience, New York, NY, USA.

Yang, C., J. Zhang. 1999. Two general methods for inverse optimization problems. *Applied Mathematics Letters* **12**(2) 69–72.

Zeleny, M. 1974. *Linear Multiobjecive Programming*, *Lecture Notes in Economics and Mathematical Systems*, vol. 95. Springer-Verlag, Berlin, Heidelberg, New York.

Zhang, J., Z. Liu. 1996. Calculating some inverse linear programming problems. *Journal of Computational and Applied Mathematics* **72**(2) 261–273.

Zhang, J., Z. Liu. 2002. A general model of some inverse combinatorial optimization problems and its solution method under l_∞ norm. *Journal of Combinatorial Optimization* **6**(2) 207–227.

Index