CoDE-SMG – Technical Report Series
ISSN 2030-6296

Published by:
CoDE-SMG, CP 210/01
Université Libre de Bruxelles
Bvd du Triomphe
1050 Ixelles, Belgium

Technical report number TR/SMG/2011-001

The Inverse $\{0, 1\}$-Knapsack Problem: Theory, Algorithms and Computational Experiments

CoDE-SMG – Technical Report Series

Julien Roland juroland@ulb.ac.be
José Rui Figueira jose.figueira@mines.inpl-nancy.fr
Yves De Smet yves.de.smet@ulb.ac.be
CoDE-SMG, Université Libre de Bruxelles, Brussels, Belgium

January 2011

Abstract

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 paper, 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.

1 Introduction

An inverse optimization problem can be 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. Inverse optimization is a very recent mathematical programming model with strong developments, and of a 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).

Recently, several approaches have been suggested for dealing with **NP**-Complete problems under the L_1 norm (e.g., the inverse integer linear programming problem by Huang (2005); Schaefer (2009), and the inverse {0,1}-knapsack problem by Huang (2005)). However, the said approaches are either computationally expensive, or fail in the search of inverse solutions. The method proposed by Huang (2005) is based on the reduction from the knapsack problem to the shortest path problem. From this reduction the author proposed to solve the inverse {0,1}-knapsack problem as the inverse shortest path problem. However, his conclusion is not valid since the network associate with the knapsack problem is a very particular one and therefore any modification on the weights should be applied with caution. In other words, the inverse {0,1}-knapsack problem can be solved as the inverse shortest path problem only with additional constraints, which was not the case in the work by Huang (2005).

Since about half a century, knapsack problems have had a strong impact on a broad array of areas, such as project selection (Petersen, 1967), capital budgeting (Lorie and Savage, 1955;

Weingartner, 1966; Weingartner and Ness, 1967), cutting stock and cargo-loading (Gilmore and Gomory, 1966). In almost all these problems, one may stress that it is usually difficult to fix the benefits in the objective function with certainty.

Consider the following project selection problem as a motivation for the inverse {0,1}-knapsack problem under the L_∞ norm in a context of uncertainty. Let $J = \{1, 2, \dots, j, \dots, n\}$ denote a set of n projects with annual profits c_j^0 and costs w_j for all $j \in J$. With an annual investment budget W, a decision maker chooses the optimal solution of this knapsack denoted by the binary-valued vector x^0 as the portfolio of the selected projects that maximize the profit value. The following year, due to economic and social changes, the profit values c_j^0 are modified. Let c_j^1 for all $j \in J$ denote the new profit values. According to the new values, the initial portfolio x^0 is no longer an optimal solution. The decision maker could ask the logical question, "Should I change my portfolio of projects?" Let us assume there is an uncertainty on each profit value c_j^1 where the exact values belong to the range $[c_j^1 - \delta, c_j^1 + \delta]$ and where δ represents an uncertainty threshold. The decision maker could agree that x^0 is still an optimal solution if some conditions were fulfilled. For instance, it would be sufficient that all the initial profit values c_j^0 belong to the range. Inverse optimization can be used to establish a more general condition as follows: let us measure the minimal adjustment of c^1 such that x^0 becomes optimum. If the minimal adjustment is less than δ, then x^0 is still an optimum. If no conditions are fulfilled, the decision maker could either choose another portfolio, or set a profit target for each project belonging to x^0.

This paper studies the inverse $\{0,1\}$-knapsack problem. 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. To our best knowledge, this is the first algorithm designed for solving the problem. 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 paper also proposes a bilevel integer linear programming problem 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 paper have been implemented and computational experiments were made on a large set of randomly generated instances. The purpose of the experiments is both to prove the feasibility of the approaches and to compare some of them.

This paper is organized as follows: in Section 2, concepts, definitions, and notation are introduced. In Section 3, a pseudo-polynomial time algorithm is presented to solve the inverse $\{0,1\}$-knapsack problem under the L_∞ distance. In Section 4, an integer linear program for solving the inverse problem is stated in the context of the L_1 distance. In Section 5, a bilevel programming approach is developed. In Section 6, computational experiments with knapsack problems under the L_∞ and the L_1 norms are presented. We conclude with remarks and some avenues for future research.

2 Concepts, Definitions and Notation

This section addresses several important concepts, their definitions and the basic notation required for the remaining sections of this paper. Let us define a linear {0,1}-combinatorial optimization problem as a set Π of instances.

Definition 2.1 (An instance of a linear {0,1}-combinatorial optimization problem)**.** *Let* $X \subseteq \Big\{x : x \in \{0,1\}^n\Big\}$ *denote a feasible set,* $J = \{1, 2, \ldots, j \ldots, n\}$ *the set of element indices, and* $c \in \mathbb{R}^n$ *a profit vector. For each solution* $x \in X$*, consider the linear expression* $f(x) = \sum_{j\in J} c_j x_j$*. A linear {0,1}-combinatorial optimization problem is a pair* (X, c) *and consists in finding* $x^* \in X$ *such that* $x^* \in \arg\max\{f(x) : x \in X\}$.

The {0,1}-Knapsack problem (KP) is a well-known classical linear {0,1}-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 denote a set composed of n items with profits $c_j \in \mathbb{N}_0 = \{0, 1, 2, \ldots\}$ and weights $w_j \in \mathbb{N}_0$, for each $j \in J$. Let $W \in \mathbb{N}_0$ denote the capacity of a knapsack. The {0,1}-Knapsack problem consists in 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 modeled as an integer linear programming problem as follows.

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

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 $c \in \mathbb{N}_0^n$, *i.e.*, (X,c). It is assumed that $\sum_{j\in J} w_j > W$ and $w_j \leqslant W$, for all $j \in J$, otherwise, the problem is obvious.

Definition 2.2 (Inverse optimization problem under the profit vector)**.** *Let* Π *denote a linear optimization problem,* $(X, c) \in \Pi$ *an instance of* Π *and* $x^0 \in X$ *a feasible solution. The associated inverse problem consists in determining a profit vector* $d^* \in \mathbb{R}^n$ *such that* x^0 *is an optimal solution of* (X, d^*) *and* $||d^* - c||_p$ *is minimum and where* $||\cdot||_p$ *is an* L_p *norm.*

The problem can be stated as a bilevel optimization problem.

$$\begin{array}{rl} \min & ||d - c||_p \\ \text{subject to:} & d^T x^* = d^T x^0 \\ & x^* \in \arg\max\{d^T x : x \in X\} \\ & d \in \mathbb{R}^n \end{array}$$

In what follows, an inverse combinatorial optimization problem under the profit vector is studied: *The inverse {0,1}-knapsack problem.*

3 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. To our best knowledge, it is the first algorithm ever designed for such a purpose.

3.1 Problem Definition

Let (X, c) denote an instance of the $\{0, 1\}$-knapsack problem and $x^0 \in X$ a feasible solution. Consider the L_∞ distance between two vectors c and d, *i.e.*, $\max\{|c_1 - d_1|, |c_2 - d_2|, \ldots, |c_j - d_j|, \ldots, |c_n - d_n|\}$. The L_∞ *inverse* $\{0, 1\}$*-knapsack problem* (IKP-∞) can be stated as follows:

$$\begin{array}{rl} d^* \in \arg\min & \max_{j\in J}\{|c_j - d_j|\} \\ \text{subject to:} & d^T x^* = d^T x^0 \\ & x^* \in \arg\max\{d^T x : x \in X\} \\ & d \in \mathbb{N}_0^n \end{array} \tag{IKP-∞}$$

IKP-∞ is a bilevel optimization problem which determines a profit vector $d^* \in \mathbb{N}_0^n$, 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^*).

3.2 Some Theoretical Results

We start by analyzing the nature of some optimal solutions of IKP-∞. Based on a partition of J defined by $J^0 = \{j \in J : x_j^0 = 0\}$ and $J^1 = \{j \in J : x_j^0 = 1\}$, 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 3.1. *There exists an optimal solution $d^* \in \mathbb{N}_0^n$ 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 $d \in \mathbb{N}_0^n$ denote any optimal solution of IKP-∞, $J^{0>} = \{j \in J^0 : d_j > c_j\}$, $J^{0\leqslant} = \{j \in J^0 : d_j \leqslant c_j\}$, $J^{1<} = \{j \in J^1 : d_j < c_j\}$, and $J^{1\geqslant} = \{j \in J^1 : d_j \geqslant c_j\}$. The constraints set of IKP-∞ implies that $d^T x^0 \geqslant d^T x$ for all feasible solutions $x \in X$. This can be written as follows:

$$\sum_{j \in J^{1\geqslant}} d_j x_j^0 + \sum_{j \in J^{1<}} d_j x_j^0 \geqslant \sum_{j \in J^{1\geqslant}} d_j x_j + \sum_{j \in J^{1<}} d_j x_j + \sum_{j \in J^{0\leqslant}} d_j x_j + \sum_{j \in J^{0>}} d_j x_j$$

by definition of $J^{1<}$ and $J^{0>}$, one obtains:

$$\sum_{j \in J^{1\geqslant}} d_j x_j^0 + \sum_{j \in J^{1<}} c_j x_j^0 \geqslant \sum_{j \in J^{1\geqslant}} d_j x_j + \sum_{j \in J^{1<}} c_j x_j + \sum_{j \in J^{0\leqslant}} d_j x_j + \sum_{j \in J^{0>}} c_j x_j \tag{1}$$

Let us define a vector d^* for all $j \in J$ as follows:

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

From this definition, it is easy to see that $\max_{j \in J}\{|c_j - d_j^*|\} \leqslant \max_{j \in J}\{|c_j - d_j|\}$ and that equation (1) implies $d^{*T} x^0 = \max\{d^{*T} x : x \in X\}$. Therefore, d^* is an optimal solution of IKP-∞, and the theorem is proved. $\square$

Let us define a vector $d^k \in \mathbb{N}_0^n$ of distance at most k from vector c with respect to the L_∞ norm.

Definition 3.2 (Vector d^k)**.** *Given a $k \in \mathbb{N}_0$, for all $j \in J$,*

$$d_j^k = \begin{cases} \max\{0, c_j - k\}, & \text{if } x_j^0 = 0, \\ c_j + k, & \text{if } x_j^0 = 1. \end{cases}$$

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

Theorem 3.3. *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 3.1, 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 $J^{1k} = J^1 \cap J^k$ and $J^{0k} = J^0 \cap J^k$. Thus, for all $x \in X$:

$$\sum_{j \in J \backslash J^k} d_j^* x_j^0 + \sum_{j \in J^{1k}} d_j^* x_j^0 \geqslant \sum_{j \in J \backslash J^k} d_j^* x_j + \sum_{j \in J^{1k}} d_j^* x_j + \sum_{j \in J^{0k}} d_j^* x_j$$

Based on the definition of d^k,

$$\sum_{j \in J \backslash J^k} d_j^k x_j^0 + \sum_{j \in J^{1k}} d_j^k x_j^0 \geqslant \sum_{j \in J \backslash J^k} d_j^k x_j + \sum_{j \in J^{1k}} d_j^k x_j + \sum_{j \in J^{0k}} d_j^* x_j .$$

Since $\sum_{j \in J^{0k}} d_j^* x_j \geqslant \sum_{j \in J^{0k}} d_j^k x_j$, one obtains:

$$\sum_{j \in J \backslash J^k} d_j^k x_j^0 + \sum_{j \in J^{1k}} d_j^k x_j^0 \geqslant \sum_{j \in J \backslash J^k} d_j^k x_j + \sum_{j \in J^{1k}} d_j^k x_j + \sum_{j \in J^{0k}} d_j^k x_j$$

Therefore, for all $x \in X$:

$$\sum_{j \in J} d_j^k x_j^0 \geqslant \sum_{j \in J} d_j^k x_j$$

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 find 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 research domain, an upper bound on the distance is provided in the following lemma.

Lemma 3.4. *Let $D_\infty \in \mathbb{N}_0$ denote the optimal solution value of IKP-∞:*

$$\begin{array}{rl} D_\infty = \min & \max_{j \in J}\{|c_j - d_j|\} \\ \text{subject to:} & d^T x^* = d^T x^0 \\ & x^* \in \arg\max\{d^T x : x \in X\} \\ & d \in \mathbb{N}_0^n \end{array}$$

Then, $D_\infty \leqslant C = \max_{j \in J}\{(1 - x_j^0) c_j\}$

Proof. It is always possible to build a vector $d \in \mathbb{N}_0^n$ with $\max_{j \in J}\{|c_j - d_j|\} = \max_{j \in J^0}\{c_j\}$ such that $d^T x^0 = \max\{d^T x : x \in X\}$. 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 $d^T x^0 \geqslant d^T x$ 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 3.5.

Theorem 3.5. *If d^k satisfies $d^{kT} x^0 = \max\{d^{kT} x : x \in X\}$, then for all $k' \in \mathbb{N}_0$, with $k' > k$, one obtains $d^{k'T} x^0 = \max\{d^{k'T} x : x \in X\}$.*

Proof. Assume there exists a solution $x \in X$ such that $d^{k'T} x > d^{k'T} x^0$. This can be written as follows:

$$\sum_{j \in J^0} d_j^{k'} x_j + \sum_{j \in J^1} d_j^{k'} x_j > \sum_{j \in J^1} d_j^{k'} x_j^0$$

Based the definition of d^k, one obtains:

$$\sum_{j \in J^0} (d_j^k - \alpha_j) x_j + \sum_{j \in J^1} (d_j^k + \alpha_j) x_j > \sum_{j \in J^1} (d_j^k + \alpha_j)$$

where $\alpha_j = |d_j^{k'} - d_j^k|$, for all $j \in J$. Consequently,

$$\sum_{j \in J} d_j^k x_j > \sum_{j \in J} d_j^k x_j^0 + \underbrace{\sum_{j \in J^0} x_j \alpha_j + \sum_{j \in J^1} (1 - x_j) \alpha_j}_{\geqslant 0} .$$

Therefore $d^{kT} x > d^{kT} x^0$, thus contradicting the hypothesis on d^k. This concludes the proof. □

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 $d \in \mathbb{N}_0^n$ such that $\max_{j \in J}\{|c_j - d_j|\} \leqslant k$, $d^T x^* = d^T x^0$ and $x^* \in \arg\max\{d^T x : x \in X\}$?

The 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 3.4, 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 bound from above by $\log_2 C$, which is polynomial in input length. Furthermore, based on Theorem 3.3, 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.

Theorem 3.6. *IKDP-∞ is **co-NP**-Complete.*

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

$\overline{\textbf{IKDP-}\boldsymbol{\infty}}$
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 feasible solution $x \in X$ such that $d^{kT} x > d^{kT} x^0$?

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 need only guess a subset of J represented by an incidence vector x and check in polynomial time that $x \in X$ and $d^{kT} x > d^{kT} x^0$.

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 $t \in \mathbb{N}_0$.
QUESTION: Is there an $x \in X$ with $c^T x \geqslant t$?

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 $c^T x \geqslant t$. Therefore, in the corresponding instance of $\overline{\text{IKDP-}\infty}$, there exists an $x \in X$ such that $d^{kT} x > d^{kT} x^0$. For any given "No" instance of KDP, $c^T x < t$ for all $x \in X$. Therefore, in the corresponding instance of $\overline{\text{IKDP-}\infty}$, there is no $x \in X$ such that $d^{kT} x > d^{kT} x^0$. 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 3.3 and 3.5. 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. □

As a corollary of this Theorem, finding a polynomial time algorithm for solving IKP-∞ is more than unlikely. However, a pseudo-polynomial time algorithm is provided in what follows, which implies that IKP-∞ is fortunately a weakly **co-NP**-Complete problem.

3.3 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 3.4, 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 $d^{kT}x^* = d^{kT}x^0$, then stop. Set $d^* \leftarrow d^k$. Otherwise, repeat the previous step with $k \leftarrow k + 1$.

The following pseudo-polynomial time algorithm (see Algorithm 1 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 1 Compute an optimal solution d^* of IKP-∞.

1: $C \leftarrow \max_{j \in J}\{(1 - x_j^0)c_j\}$;
2: OPT $\leftarrow KP(X, c)$;
3: **for all** $(k = 1$ to $C)$ **and** (OPT $> d^{kT}x^0$) **do**
4:     **for all** $(j = 1$ to $n)$ **do**
5:         **if** $(x_j^0 = 0)$ **then**
6:             $d_j^k \leftarrow \max\{0, c_j - k\}$;
7:         **else if** $(x_j^0 = 1)$ **then**
8:             $d_j^k \leftarrow c_j + k$;
9:         **end if**
10:     **end for**
11:     OPT $\leftarrow KP(X, d^k)$;
12: **end for**
13: $d^* \leftarrow d^k$;

Theorem 3.7. *Algorithm (1) determines a vector $d^* \in \mathbb{N}_0^n$ that is an optimal solution of IKP-∞.*

Proof. Directly deduced from Lemma 3.4 and Theorem 3.3. □

Theorem 3.8. *Algorithm (1) runs in $O(nWC)$.*

Proof. 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*. *Sketch of the algorithm.* Based on Lemma 3.4, 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 3.3 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 2 for a pseudo-code of this procedure. Finally, note that the correctness of the algorithm relies on Theorem 3.5, which expresses that the value of k can be increased without altering the optimality of x^0 with respect to vector d^k.

4 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

Algorithm 2 Compute an optimal solution d^* of IKP-∞.

$a \leftarrow 0$;
$b \leftarrow C$;
while $a \neq b$ **do**
  $k \leftarrow a + \lfloor (b-a)/2 \rfloor$;
  for all $(j = 1 \text{ to } n)$ **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
  $\text{OPT} \leftarrow KP(X, d^k)$;
  if $OPT = d^{kT} x^0$ **then**
    $b \leftarrow k$;
  else
    $a \leftarrow k + 1$;
  end if
end while
$d^* \leftarrow d^k$;

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 pseudo-polynomial algorithm proposed by Huang (2005) fails in the search of inverse solutions and the integer linear programming formulation of Schaefer (2009) suggested to solve inverse integer programming problems is defined by an exponential number of constraints.

4.1 Problem Definition

Let $x^0 \in X$ denote a feasible solution. Consider the L_1 distance between two vectors c and d, *i.e.*, $|c_1 - d_1| + |c_2 - d_2| + \ldots + |c_j - d_j| + \ldots + |c_n - d_n|$. The L_1 *inverse* $\{0,1\}$*-knapsack problem* (IKP-1) can be stated as follows:

$$\begin{array}{rl} \min & \sum_{j \in J} |c_j - d_j| \\ \text{subject to:} & d^T x^* = d^T x^0 \\ & x^* \in \arg\max\{d^T x : x \in X\} \\ & d \in \mathbb{N}_0^n \end{array} \qquad \text{(IKP-1)}$$

IKP-1 is a bilevel optimization problem that determines a profit vector $d^* \in \mathbb{N}_0^n$, which minimizes the L_1 distance with respect to c and such that x^0 is an optimal solution of the modified knapsack problem (X, d^*).

4.2 Some Theoretical Results

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

Lemma 4.1. *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*

$$||d^* - c||_1 = \sum_{j \in J} |d_j^* - c_j| \geqslant (c^T x' - c^T x^0)$$

Proof.

$$
\begin{aligned}
||d^* - c||_1 &\geqslant (d^{*T} - c^T)(x^0 - x') \\
&= d^{*T}x^0 - d^{*T}x' - c^T x^0 + c^T x' \\
&= (d^{*T}x^0 - d^{*T}x') + (c^T x' - c^T x^0) \\
&\geqslant (c^T x' - c^T x^0),
\end{aligned}
$$

where the first inequality holds because both x^0 and x' are 0-1 vectors and the second inequality results from $d^{*T}x^0 \geqslant d^{*T}x'$. 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:

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

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

The following counter example shows this negative result:

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

If $x^0 = (0, 0, 0)$ is considered, then $(c^T x^* - c^T x^0) = 1$ 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 4.2. *Let $D_1 \in \mathbb{N}_0$ denote the optimal solution value of IKP-1:*

$$
\begin{array}{rl}
D_1 = \min & \sum_{j \in J} |c_j - d_j| \\
\text{subject to:} & d^T x^* = d^T x^0 \\
 & x^* \in \arg\max\{d^T x : x \in X\} \\
 & d \in \mathbb{N}_0^n
\end{array}
$$

Then, $D_1 \leqslant \sum_{j \in J} \{(1 - x_j^0) c_j\}$.

Proof. Proved by using the same arguments as in Lemma 3.4. □

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 $d \in \mathbb{N}_0^n$ so that, $\sum_{j \in J} |c_j - d_j| \leqslant k$, $d^T x^* = d^T x^0$ and $x^* \in \arg\max\{d^T x : x \in X\}$?

For the same reasons as in IKP-∞, the IKP-1 can be solved by using a *binary search* for the optimal solution value through a certain number of calls to IKDP-1.

Theorem 4.3. *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 $j = 1, \ldots, n$, $x_{n+1}^0 = 1$, 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 $d^T x^* = d^T x^0$ and $x^* \in \arg\max\{d^T x : x \in X\}$ 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 $d^T x^* = d^T x^0$ and $x^* \in \arg\max\{d^T x : x \in X\}$ are not satisfied because $d^T x^0 = t - 1$ and by hypothesis, there exists an $x \in X$ such that $d^T x = t$. Consequently, IKDP-1 is **co-NP**-Hard and the theorem is proved. □

However, IKDP-1 has not be 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 4.4. *IKDP-1 belongs to* $\mathbf{NP}^{KDP}$

Proof. A nondeterministic polynomial time oracle Turing machine with KDP oracle can check whether $d \in \mathbb{N}_0^n$ satisfies the conditions $\sum_{j \in J} |c_j - d_j| \leq k$, $d^T x^* = d^T x^0$, and $x^* \in \arg\max\{d^T x : x \in X\}$. 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 paper.

4.3 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 (2).

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

By minimizing $g_n(W)$ over this set of constraint, it is easy to see that one obtains the optimal value of the knapsack problem. Consider the following integer linear programming model (Problem 3).

$$\begin{array}{rl} \min & \sum_{j \in J} \delta_j \\ \text{subject to:} & \delta_j \geqslant d_j - c_j, \; j \in J \\ & \delta_j \geqslant c_j - d_j, \; j \in J \\ & \sum_{j \in J} d_j x_j^0 \geqslant g_n(W) \\ & \text{The set of constraints (2) on } g_n(W) \\ & d \in \mathbb{N}_0^n \\ & \delta_j \in \mathbb{N}_0, \; j \in J \end{array} \tag{3}$$

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

Let us establish the exactness of this formulation for solving IKP-1. The following lemma expresses that x^0 is an optimal solution for all vectors $d \in \mathbb{N}_0^n$ satisfying the set of constraints described in Problem 3.

Lemma 4.5. *For all vectors $d \in \mathbb{N}_0^n$ satisfying the set of constraints described in Problem 3, $d^T x^0 = \max\{d^T x : x \in X\}$.*

Proof. By definition, $g_n(W) \geqslant \max\{d^T x : x \in X\}$ and the set of constraints implies $d^T x^0 \geqslant g_n(W)$. Then,

$$d^T x^0 \geqslant g_n(W) \geqslant \max\{d^T x : x \in X\} \geqslant d^T x^0,$$

where the last inequality results from the fact that $x^0 \in X$. This concludes the proof. □

It is important to prove that all vectors leading to the optimality of x^0 satisfy the set of constraints described in Problem 3. This is established by the following lemma.

Lemma 4.6. *For all vectors $d \in \mathbb{N}_0^n$ with $d^T x^0 = \max\{d^T x : x \in X\}$, there exists a $g_n(W) \in \mathbb{N}_0$ such that $\sum_{j \in J} d_j x_j^0 \geqslant g_n(W)$.*

Proof. For all vectors $d \in \mathbb{N}_0^n$ with $d^T x^0 = \max\{d^T x : x \in X\}$, one can build a $g_n(W) \in \mathbb{N}_0$ such that $\sum_{j \in J} d_j x_j^0 = g_n(W)$. This results from the definition of $g_n(W)$. □

Theorem 4.7. *Problem 3 determines a vector $d^* \in \mathbb{N}^0$ with $d^{*T} x^0 = \max\{d^{*T} x : x \in X\}$ and such that there is no $d' \in \mathbb{N}^0$ 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. This directly results from Lemmas 4.5 and 4.6. □

This establishes the exactness of this formulation for solving IKP-1.

If the integrality constraint on vector d is removed, the integer program (3) becomes a linear programming model. Then, Problem 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 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.

5 A Bilevel Programing 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.

5.1 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:

$$\begin{array}{rl} \min & \delta \\ \text{subject to:} & \delta \geqslant c_j - d_j,\ j \in J \\ & \delta \geqslant d_j - c_j,\ j \in J \\ & d^T x^* = d^T x^0 \\ & x^* \in \arg\max\{d^T x : x \in X\} \\ & d \in \mathbb{N}_0^n \\ & \delta \in \mathbb{R} \end{array}$$

The non-linear vector product $d^T x^*$ 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 is the upper bound on the the value of d_j^*, with $j \in J$. This upper bound is provided in the following lemma.

Lemma 5.1. *If d^* is an optimal solution of IKP-1, then for all $j \in J$, $d_j \leqslant M = 2\max_{j\in J}\{c_j\}$.*

Proof. Assume there exists an $i \in J$, such that $d_i^* > 2\max_{j\in J}\{(1-x_j^0)c_j\}$. Therefore, $\max_{j\in J}\{|c_j - d_j^*|\} > \max_{j\in J}\{(1-x_j^0)c_j\}$, which contradicts Lemma 3.4, and this lemma is proved. □

Finally, replace $d^T x$ 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:

$$\begin{array}{rlll} \min & \delta & & \\ \text{subject to:} & \delta \geqslant d_j - c_j,\ j \in J & & \\ & \delta \geqslant c_j - d_j,\ j \in J & & \\ & \sum_{j\in J} e_j^* = d^T x^0 & & \\ & d \in \mathbb{N}_0^n & & \\ & \delta \in \mathbb{R} & & \\ & e^* \in \arg\max \sum_{j\in J} e_j & & \\ & \quad\quad \text{subject to:} & e_j \leqslant d_j,\ j \in J & \\ & & e_j \leqslant x_j M,\ j \in J & \\ & & e \in \mathbb{R}_+^n & \\ & & x \in X & \end{array} \tag{4}$$

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

Lemma 5.2. *If $e \in \mathbb{R}_+^n$ and $x \in X$ denote a feasible solution of the lower-level problem of the bilevel optimization problem (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 5.3. *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 (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 5.4. *Let $e^* \in \mathbb{R}^n_+$ denote an optimal solution of the lower level problem of the bilevel optimization problem (4). Then, $\sum_{j \in J} e^*_j = \max\{d^T x : x \in X\}$.*

Proof. This results directly from Lemmas 5.2 and 5.3. □

5.2 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 $\sum_{j \in J} e_j = d^T x^0$ into the objective function. With $L = C + 1$, the following equivalent problem is obtained.

$$\begin{array}{rll} \min & \delta + \left(\sum_{j \in J} e_j - d^T x^0 \right) L & \\ \text{subject to:} & \delta \geqslant d_j - c_j, \ j \in J & \\ & \delta \geqslant c_j - d_j, \ j \in J & \\ & d \in \mathbb{N}^n_0 & \\ & \delta \in \mathbb{R} & \\ & e \in \arg\max \sum_{j \in J} e_j & \\ & \qquad\qquad \text{subject to:} \quad e_j \leqslant d_j, \ j \in J & \\ & \qquad\qquad\qquad\qquad\qquad e_j \leqslant x_j M, \ j \in J & \\ & \qquad\qquad\qquad\qquad\qquad e \in \mathbb{R}^n_+ & \\ & \qquad\qquad\qquad\qquad\qquad x \in X & \end{array} \tag{5}$$

Based on Lemma 3.4, it is known that $\delta \leqslant C$. Therefore, a solution of Problem (5) with $d^T x^0 < \sum_{j \in J} e_j$ is not an optimum, because in this case, $\delta + \left(\sum_{j \in J} e_j - d^T x^0 \right) L \geqslant L > C$.

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 Computational Experiments

The purpose of this section is to report the behavior of Algorithm 1, Algorithm 2 and the integer programming model of Problem (3) on several sets of randomly generated instances. This helps compare the approaches proposed in this paper and measure their practical application in terms of performance. The design of the experiments is inspired by the frameworks used in Martello and Toth (1990) and in Pisinger (1995).

6.1 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 and Stutz, 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 instance, 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.

Algorithms were implemented in the C++ programming language. Algorithms 1 and 2 integrate the algorithm for the {0,1}-knapsack problem proposed by Pisinger (1997) (The implementation of this algorithm is available at `http://www.diku.dk/~pisinger/codes.html`) and the integer linear program (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 operation system. This make 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.

6.2 Statistical Analysis

For the analysis of Algorithms 1 and 2, 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.

Algorithm 2 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. The application of the binary search in Algorithm 2 is significant because with $n = 10^4$, $R = 10^4$ and $P = 0.5$, the average computation time of Algorithm 1 is 539.98 seconds with a 95% confidence interval given by $[364.41, 715.54]$. Let us point out that the performances of these algorithms are strongly linked to the embedded knapsack solver. Therefore, the use of another solver could either increase or decrease significantly the computation time.

Let us compare the performance of Algorithms 1 and 2 more precisely by computing the pairwise differences in running times defined by $\Delta_i = \delta_i^1 - \delta_i^2$, where δ_i^1 is the computation time of Algorithm 1 on the i-th instance and δ_i^2 is the computation time of Algorithm 2 on the same instance. Let M denote the median of the population of pairwise differences. Then $M > 0$ implies that Algorithm 1 is likely to take longer, whereas $M < 0$ implies that Algorithm 2 takes longer. To compare the two algorithms the test is $H_0 : M = 0$ versus $H_a : M \neq 0$ (see, for example, Coffin and Saltzman, 2000, for more details).

Consider a population of 30 strongly correlated instances with $n = 10^4$, $R = 10^4$, and $P = 0.5$. By the use of the sign test with a significance level $\alpha = 0.05$, hypothesis H_0 is rejected and one may conclude that the algorithms perform differently. A 95% confidence interval for M (based on the binomial distribution) is $[177.09, 767.81]$, indicating that Algorithm 1 should take between 177.09 and 767.81 seconds more that Algorithm 2 for solving the same instance. Consider also a population of 30 strongly correlated instances with $n = 10^3$, $R = 10^4$, and $P = 0.5$. Hypothesis H_0 is also rejected and a 95% confidence interval for M is $[9.19, 22.58]$. Finally, consider a population of 30 strongly correlated instances with $n = 10^3$, $R = 10^3$, and $P = 0.5$. Hypothesis H_0 is also rejected and a 95% confidence interval for M is $[0.1, 0.57]$.

A multiple regression model of running time can be used for an empirical analysis of average complexity (see, for example, Vanderbei, 1996; Coffin and Saltzman, 2000). Since the complexity of Algorithm 1 is $O(nWC)$, the running time can be modeled by $T = nWC$. This model is a log-linear one:

$$\log_e T = \log_e n + \log_e W + \log_e C + \epsilon.$$

Consider strongly correlated instances of Type 2 with $R = 10^4$, $n \in \{10^2, 5 \times 10^2, 10^3, 5 \times 10^3, 10^4, 5 \times 10^4, 10^5\}$ and $P \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$. For each couple (n, P), 30 instances were generated. The fitted regression model is obtained for this set by using the well-known least-squares method is as follows:

$$\log_e T \approx -1.721 + 0.553 \log_e n + 0.9185 \log_e W - 1.476 \log_e C$$

The adjusted coefficient of multiple determination is 0.839. Consequently 83.9% of the variability of T is explained by the variables n, W and C. It is worth mentioning that influence of variables n, W and C can also be explained by the choice of the embedded knapsack solver.

The same can be done for Algorithm 2:

$$\log_e T \approx -94.430 + 3.557 \log_e n - 2.271 \log_e W + 39.583 \log_e \log_2 C$$

The adjusted coefficient of multiple determination is 0.94. Here the influence of $\log_2 C$ is very important, because the main loop runs exactly $\log_2 C$ times and at each iteration the instance is strongly modified therefore influencing the performance of the knapsack solver.

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 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 $[3.93, 8.97]$.

For more details on the results of the computational experiments, the reader may consult the Appendix.

7 Conclusion

In this paper, 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 easily 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.

A Computational Results

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

	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	0.02	0.02	0.00	0.09	0.15	0.13	0.00	0.41	0.61	0.53	0.02	1.97
500	0.22	0.25	0.00	0.82	2.83	1.78	0.14	6.23	15.42	10.74	0.75	47.81
1000	0.45	0.47	0.00	1.83	8.84	6.90	0.55	27.20	24.95	22.42	0.21	74.63
5000	3.16	2.98	0.01	10.84	51.04	51.38	0.01	178.91	185.86	159.35	11.41	585.12
10000	7.66	7.91	0.02	22.23	151.15	132.95	5.71	443.04	541.53	472.01	17.20	1519.45

Table 3: Impact of varying the number of variables n and data range R on performance of Algorithm 1 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	0.27	0.14	0.00	0.46	4.45	2.28	0.31	8.05	15.11	8.26	0.71	29.27
0.2	0.33	0.24	0.00	0.71	6.20	3.64	0.47	13.08	21.16	12.59	1.49	46.03
0.3	0.32	0.17	0.00	0.54	4.69	2.83	0.02	10.66	16.92	13.37	0.33	43.53
0.4	0.33	0.28	0.05	1.23	6.02	4.81	0.44	16.28	26.20	17.31	1.54	68.42
0.5	0.45	0.47	0.00	1.83	8.84	6.90	0.55	27.20	24.95	22.42	0.21	74.63
0.6	0.72	0.67	0.00	2.06	9.53	10.08	0.58	33.60	25.03	22.59	0.62	96.51
0.7	0.60	0.65	0.00	2.04	18.26	16.19	0.08	44.54	45.01	52.66	0.18	171.93
0.8	0.70	0.67	0.00	2.40	15.72	15.76	0.09	47.47	54.56	54.28	0.10	180.19
0.9	0.40	0.45	0.00	1.63	13.12	13.16	0.02	36.09	50.58	52.72	0.58	178.21

Table 4: Impact of varying the percentage P and data range R on performance of Algorithm 1 with a group of strongly correlated instances of Type 2 and $n = 10^3$.

	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 5: Impact of varying the number of variables n and data range R on performance of the integer linear programming problem 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 6: Impact of varying the percentage P and data range R on performance of the integer linear programming problem 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 7: Impact of varying the number of variables n and data range R on performance of the integer linear programming problem 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 8: Impact of varying the percentage P and data range R on performance of the integer linear programming problem 3 with a group of strongly correlated instances of Type 2 and $n = 10$.

Acknowledgments

The autors would like to thank David Pisinger for providing a copy of the Minknap algorithm and the knapsack generator, and IBM for allowing us free access to the IBM ILOG CPLEX Optimizer. This research partially benefited from the COST Action IC0602 on "Algorithmic Decision Theory".

References

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

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

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

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

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

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

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

C. C. Petersen, Computational experience with variants of the balas algorithm applied to the selection of R&D projects, Management Science 13 (1967) 736–750.

J. H. Lorie, L. J. Savage, Three problems in rationing capital, The Journal of Business 28 (1955) 229–239.

H. M. Weingartner, Capital budgeting of interrelated projects: Survey and synthesis, Management Science 12 (1966) 485–516.

H. M. Weingartner, D. N. Ness, Methods for the solution of the multidimensional 0/1 knapsack problem, Operations Research 15 (1967) 83–103.

P. C. Gilmore, R. E. Gomory, The theory and computation of knapsack functions, Operations Research 14 (1966) 1045–1074.

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

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

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

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

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

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

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

D. Pisinger, Algorithms for Knapsack Problems, Ph.D. thesis, Department of Computer Science, University of Copenhagen, Denmark, 1995.

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

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

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

R. J. Vanderbei, Linear programming: foundations and extensions, Kluwer Academic Publishers, Boston, USA, 1996.