The Knapsack Problem With Gaussian Weights *

B. Fortz · M. Labbé · F. Louveaux · M. Poss

Received: date / Accepted: date

Abstract The main difficulty in two-stage stochastic programming with real recourse is the number of scenarios to consider, resulting in a huge number of variables and constraints. In this work, we overcome this difficulty for the knapsack problem with penalty recourse by considering mainly Gaussian random variables. Using their summation property, we can simplify the problem and avoid performing multiple integrations for evaluating the objective. Then, complexity results are given for two stochastic versions of the subset sum problem, and for the general problem with constant weights and capacity uniformly distributed. Finally, computational experiments for Gaussian weights prove this approach to be efficient.

Keywords Stochastic programming · Knapsack problem · Complexity · Mixed integer non linear programming.

* This research is supported by an Actions de Recherche Concertées (ARC) project of the Communauté française de Belgique. Michael Poss is a research fellow of the Fonds pour la Formation à la Recherche dans l’Industrie et dans l’Agriculture (FRIA).

B. Fortz
Department of Computer Science, Faculté des Sciences, Université Libre de Bruxelles, Brussels, Belgium.
CORE, Université catholique de Louvain, Louvain-la-Neuve, Belgium.
E-mail: bfortz@ulb.ac.be

M. Labbé
Department of Computer Science, Faculté des Sciences, Université Libre de Bruxelles, Brussels, Belgium.
E-mail: mlabbe@ulb.ac.be

F. Louveaux
Louvain School of Management, FUNDP, University of Namur, Namur, Belgium.
E-mail: flouveaux@fundp.ac.be

M. Poss
Department of Computer Science, Faculté des Sciences, Université Libre de Bruxelles, Brussels, Belgium.
E-mail: mposs@ulb.ac.be
1 Introduction

The classical knapsack problem aims to find a subset of $n$ items not exceeding the capacity $a_0$ of the knapsack and maximizing the total profit. Each item $i \in N$ has a profit $p_i$ and a weight $a_i$. Our two-stage stochastic version of the problem considers that weights $a_i(\omega)$ and capacity $a_0(\omega)$ are continuous random variables. The subset of items is chosen first, then any real amount $y(\omega)$ of additional capacity can be bought at the unitary price of $K$, depending on the scenario $\omega$. The objective function aims to maximize the profit of the chosen subset minus the expected cost of the additional capacity:

$$\max \sum_{i \in N} p_i x_i - K \mathbb{E}_a[y(\omega)]$$

s.t. $\sum_{i \in N} a_i(\omega)x_i \leq a_0(\omega) + y(\omega) \quad \forall \omega \in \Omega$ \hspace{1cm} (1)

where $\Omega$ is the set of all scenarios and $\mathbb{E}_a[y(\omega)] = \int_A f_a(z)y(z)dz$ where $f_a : \mathbb{R}^{n+1} \rightarrow \mathbb{R}_+$ is the density function of the random vector $a : \Omega \rightarrow A \subset \mathbb{R}^{n+1}$. Because constraints (1) are satisfied at equality in any optimal solution with $y(\omega) > 0$, the variables $y(\omega)$ can be replaced by $\max(0, \sum a_i(\omega)x_i - a_0(\omega))$, resulting in the following formulation:

$$\max_{x \in \{0,1\}^n} \sum_{i \in N} p_i x_i - K \mathbb{E}_a \left[ \max \left( 0, \sum_{i \in N} a_i(\omega)x_i - a_0(\omega) \right) \right] \hspace{1cm} (2)$$

This problem can be thought of as the following resource allocation problem [12]. A decision maker has to choose a subset of $n$ known alternative projects to take on. For this purpose a known quantity $a_0$ of relatively low-cost resource is available to be allocated. Any additional amount of resource required can be obtained at a known incremental cost of $K$ per unit of resource. The amount $a_i$ of resource required by each project is not known at the time the decision has to be made, but we assume that the decision maker has an estimate of the probability distribution of those $a_i$. Finally, each project $i$ has an expected reward of $p_i$.

Besides being of practical application, the expected term in the objective makes the stochastic model worth to be studied. Indeed, a well known simplification when dealing with stochastic models is to replace random parameters by their means and to solve the resulting deterministic model. However, the term

$$\mathbb{E}_a \left[ \max \left( 0, \sum_{i \in N} a_i(\omega)x_i - a_0(\omega) \right) \right]$$

implies that two subsets $N_1$ and $N_2$ having equal means but very different variances, can have very different expected cost. This is not taken into account by the deterministic model considering only means.

Problem (1) has been first formulated by Barnhart and Cohn [6] who consider that $a_0$ is given and that $a_i$ follow Gaussian variables. They derive basic properties and propose a simple branch-and-bound algorithm that they test on an example with 15 variables. Recently, Kosuch and Lisser [15] use a stochastic gradient method to solve (1) with Gaussian variables. They solve the problem up to 150 variables in at most two hours. Kleywegt et al. [12] work on a similar model with discrete random
variables to test their sample average approximation method. Other ways of considering uncertainties in the parameters of the knapsack problem include chance-constrained knapsack [14, 20], robust knapsack [13], and dynamic knapsack [11], among others.

Even though problem (2) is non linear, it is unconstrained and concave for most random vectors \(a = (a_0, a_1, \ldots, a_n)\):

**Theorem 1** ([4]) For a maximization stochastic program with fixed recourse where a has finite second moments, \(Q(x)\) is a Lipschitzian concave function on \(K_2 = \{x|Q(x) < \infty\}\).

The recourse function \(Q\) is the expected optimal value of the second stage problem. Fixed recourse means that the constraint coefficients of the recourse variables are constant. Coming back to (1), the recourse function is defined by

\[
Q(x) = -K \mathbb{E}_a \left[ \min_{y(\omega) \geq 0} \left\{ y(\omega) \text{ s.t. } \sum_{i \in N} a_i(\omega)x_i \leq a_0(\omega) + y(\omega) \right\} \right],
\]

and the only recourse variable is \(y\) with a coefficient equal to 1; moreover \(K_2 = \mathbb{R}^n\). Hence we can think of using convex non-linear mixed integer techniques to tackle the problem. For instance outer-approximation type algorithms [7] approach the objective by a set of tangent planes. However to compute the coefficients of those planes, we need to evaluate many times the recourse function

\[
Q(x) = -K \mathbb{E}_a \left[ \max_{0} \left( 0, \sum_{i \in N} a_i(\omega)x_i - a_0(\omega) \right) \right] = -K \int_A f_a(a_0, a_1, \ldots, a_n) \max \left( 0, \sum_{i \in N} a_i x_i - a_0 \right) da_0 da_1 \ldots da_n, \tag{3}
\]

namely to evaluate an integral of \(n\) variables. To avoid the evaluation of those integrals, most papers on two stages stochastic problems replace the continuous random vector, in this case \(a\), by a discrete approximation. This results in solving large scale problems, with a particular structure well suited for Benders’ decomposition methods, also called L-shaped [16]. Another trend uses Monte Carlo sampling to generate a few scenarios instead of considering all of them [12, 25]. The book of Birge and Louveaux [4] gives a good introduction to stochastic programming methods.

The contributions of this paper are twofold. First we give complexity results for three groups of problems:

- Stochastic subset sum with weights and capacity distributed according to Gaussian random variables.
- Stochastic subset sum with known weights and a random capacity.
- Stochastic knapsack with known weights and uniformly distributed capacity.

Then, we show that the NP/NLP algorithm of Quesada and Grossman [23] allows us to solve efficiently the general problem with Gaussian random variables and provide computational results. The implemented algorithm is able to solve in less than a minute problems involving up to a few thousands of variables. Discretizing the random variable would result in a huge number of scenarios, making the objective impossible to compute exactly. For instance, if we consider only 100 different items whose weight can take 3 different values (weights are assumed independently distributed), there are already \(3^{100} \sim 5 \cdot 10^{47}\) scenarios !
The paper is organized as follows. In the next section, we assume that weights follow Gaussian random variables. This allows us to simplify greatly the objective function and to compute the complexity of the "stochastic subset sum", defined in Section 2.2. In Section 3, we derive two complexity results when the capacity is the only random variable. We introduce briefly a non linear integer optimization technique in Section 4, known as LP/NLP algorithm [1]. We present computational results in the last section.

In the rest of this paper, we use \( B = \{0,1\}^n \) and \( \overline{B} = [0,1]^n \); the summation \( \sum \) refers to the sum over set \( N \) unless stated otherwise.

2 Gaussian weights and capacity

2.1 General Problem

In the subsequent section, we assume that weights \( a_i, i \in N \), and the capacity \( a_0 \), follow Gaussian variables independently distributed of parameters \( \mu_i \) and \( \sigma_i \). This is motivated by their summation property: if \( X_1, \ldots, X_n \) are independent Gaussians of mean \( \mu_i \) and variance \( \sigma_i^2 \), and \( t_i \) are real numbers, then \( Y := \sum t_i X_i \sim \mathcal{N}(\mu, \sigma^2) \), with \( \mu = \sum t_i \mu_i \) and \( \sigma^2 = \sum t_i^2 \sigma_i^2 \). Moreover, Gaussians are often used to represent the error made on estimations of parameters for many physical and economical problems.

Recall that the main difficulty of (2) is to evaluate \( Q \) defined by (3). However, using the summation property, the sum

\[
\sum_{i \in N} a_i(\omega)x_i - a_0(\omega)
\]

follows a Gaussian distribution with mean \( \mu(x) = \sum \mu_i x_i - \mu_0 \) and variance \( \sigma^2(x) = \sum \sigma_i^2 x_i^2 + \sigma_0^2 \). Then, we can express \( Q \) as follows [6]:

\[
Q(x) = -K \left\{ \sigma(x)f\left(\frac{-\mu(x)}{\sigma(x)}\right) + \mu(x)G\left(\frac{-\mu(x)}{\sigma(x)}\right) \right\},
\]

(4)

where \( f \) is the density function of \( \mathcal{N}(0,1) \), \( G = 1 - F \) and \( F \) is the cumulative distribution of \( \mathcal{N}(0,1) \). Note that because Gaussians have finite second moments, Theorem 1 implies that \( Q(x) \) is concave. Finally, the optimization problem (1) for Gaussian weights and capacity can be reformulated as maximizing a concave function of binary variables:

\[
\max_{x \in B} \sum_{i \in N} p_i x_i - K \left\{ \sigma(x)f\left(\frac{-\mu(x)}{\sigma(x)}\right) + \mu(x)G\left(\frac{-\mu(x)}{\sigma(x)}\right) \right\}.
\]

(5)

We show in Section 5 how a simple LP/NLP algorithm can efficiently optimize this function.

2.2 Stochastic Subset Sum

A well known particular case of the deterministic knapsack problem is the subset sum problem, where we consider that weight \( a_i \) is equal to profit \( p_i \) for each item \( i \in N \). Even though weakly NP-hard to solve, adapted algorithms can have much better behavior for this problem than for the general knapsack.
To the best of our knowledge, no stochastic version of the subset sum has been addressed in the literature so far. To define the stochastic subset sum, we replace the deterministic constraint \( a = p \) by:

- \( \mathbb{E}[a] = p \),
- \( \text{Var}[a] = \lambda p \) for some \( \lambda \geq 0 \),

where \( \mathbb{E}[v] = (\mathbb{E}[v_1], \ldots, \mathbb{E}[v_n]) \) and \( \text{Var}[v] = (\text{Var}[v_1], \ldots, \text{Var}[v_n]) \) for any random vector \( v \). The constraint \( \mathbb{E}[a] = p \) is the direct extension of \( a = p \). Then, to enforce the link between \( p \) and \( a \) we also limit variations of \( a \), imposing \( \text{Var}[a] = \lambda p \). Note that the case \( \lambda = 0 \) results in a deterministic knapsack, where additional capacity can be purchased at the incremental cost of \( K \) per unit, see problem (8).

This definition imposes the following restrictions on problem (5): for each item \( i \in N \), profit is equal to \( \mu_i \), weight \( a_i \) follows \( N(\mu_i, \lambda \mu_i) \), and capacity \( a_0 \) follows \( N(\mu_0, \lambda \mu_0) \), for some \( \lambda \geq 0 \).

**Theorem 2** Consider the problem

\[
\max_{x \in B} \sum_{i \in N} \mu_i x_i - K \mathbb{E}_a \left[ \max \left( 0, \sum_{i \in N} a_i(\omega)x_i - a_0(\omega) \right) \right],
\]

where \( a_i \sim N(\mu_i, \lambda \mu_i), 0 \leq \lambda \leq 1 \), \( \mu \) is an integer vector and \( \mu_0 > 0 \). Problem (6) is weakly NP-hard, as it can be solved by a pseudo-polynomial algorithm in \( O(n \sum \mu_i) \).

The proof consists in two steps: first we show in Proposition 1 that the deterministic problem is a particular case of the stochastic one. Then we prove that the solution to (6) follows from the solution to two subset sum problems plus a dichotomic search.

**Proposition 1** The problem (6) is at least as hard as its deterministic counterpart.

**Proof** Consider the problem

\[
\max_{x \in B} \left\{ \sum_{i \in N} \mu_i x_i \text{ s.t. } \sum_{i \in N} \mu_i x_i \leq \mu_0 \right\}.
\]

Now, consider an instance of (6) with \( \lambda = 0 \) so that \( \sigma_i = 0 \), for each \( 0 \leq i \leq n \), that is

\[
\max_{x \in B, y \geq 0} \left\{ \sum_{i \in N} \mu_i x_i - K y \text{ s.t. } \sum_{i \in N} \mu_i x_i \leq \mu_0 + y \right\}.
\]

Because variances are equal to zero, there is only one scenario in \( \Omega \) so that \( y := \mathbb{E}[y(\omega)] \).

We show next that for a good choice of parameter \( K \), \( y^* = 0 \) in any optimal solution \((x^*, y^*)\) to (8), so that \( x^* \) is an optimal solution to (7). Hence (7) is a particular case of (8).

Consider \( K > \max_{i \in N} \mu_i \). Assume there exists an optimal solution \((x^*, y^*)\) to (8) with \( y^* \geq 1 \) (recall \( \mu_i \) are integer). Then choosing any \( i \in N \) with \( x^*_i = 1 \), we define \( \tilde{x}_j = x^*_j \) for \( i \neq j \in N \), \( \tilde{x}_i = 0 \) and \( \tilde{y} = \max(0, y^* - \mu_i) \). Defining \( Z(x, y) = \sum \mu_i x_i - K y \), we have that

\[
Z(\tilde{x}, \tilde{y}) = Z(x^*, y^*) + K(y^* - \tilde{y}) - \mu_i \geq Z(x^*, y^*) + K - \mu_i > Z(x^*, y^*)
\]

so that \((x^*, y^*)\) is not optimal to (8).

\(\square\)
Proposition 2 The problem (6) can be solved in $O(n \sum \mu_i)$. 

The rest of the section aims to prove this proposition. First of all, consider the auxiliary function $R(x) = -K \mathbb{E}[\max(0, a(\omega))]$, with $a \sim N(\mu(x), \sigma^2(x))$, $\mu(x) = \sum \mu_i x_i$ as before and $\sigma^2(x) = \sum \sigma_i^2 x_i + \sigma_0^2$. Note that for each $i \in N$, $x_i = x_i$ when $x \in B$ so that $\sigma(x) = \sigma(x)$ when $x \in B$.

Lemma 1 Define $Z_Q(x) = \sum \mu_i x_i + Q(x)$ and $Z_R(x) = \sum \mu_i x_i + R(x)$. The functions $Z_Q(x)$ and $Z_R(x)$ coincide on $B$, so that $\max_B Z_Q(x) = \max_B Z_R(x)$.

From now on we focus on the maximization of $Z_R$. The following property justifies the use of the function $R$:

Lemma 2 The function $Z_R : \mathcal{B} \to \mathbb{R}$ only depends on the variable $y = \sum \mu_i x_i$: $Z_R(x) = \tilde{Z}(y)$, where $\tilde{Z} : [0, \sum \mu_i] \to \mathbb{R}$.

Proof By definition of $y$, $Z_R$ can be rewritten as $y + R(x)$. Let us now prove that $R(x)$ only depends on $y$. As we did for $Q$ in (4), $R$ can be rewritten as

$$R(x) = -K \left\{ \hat{\sigma}(x) \left( -\frac{\mu(x)}{\sigma(x)} \right) + \mu(x) G \left( -\frac{\mu(x)}{\sigma(x)} \right) \right\}.$$ 

(9)

We see that $\mu(x) = \sum \mu_i x_i = y$ and $\hat{\sigma}(x) = \sqrt{\sum \sigma_i^2 x_i} = \sqrt{\lambda y}$. \hfill $\square$

Now, $R(x) = Q(x)$ only when $x \in B$; when $x \in \mathcal{B}/B$ these functions may be different. In particular, neither $Z_R$ nor $\tilde{Z}$ inherit from the concavity of Theorem 1. However, we can prove the result analytically for $\tilde{Z}$:

Lemma 3 If $0 < \lambda \leq 1$ and $\mu > 0$, the function $\tilde{Z}$ is concave for all $y \geq 0$.

Proof Computing the second derivative of $\tilde{Z}$, we get:

$$\tilde{Z}''(y) = -K \left( \frac{\lambda^2 (y + \mu_0 + 2 \sigma_0^2)^2 - y - \sigma_0^2 e^{-\frac{(y-\mu_0)^2}{2 \sigma_0^2}}}{8 \lambda \sqrt{y + \sigma_0^2}} \right).$$

This is always negative when $\lambda \leq 1$ and $y + \mu_0 + 2 \sigma_0^2 \geq 1$. \hfill $\square$

Recalling that $\sigma^2 = \lambda \mu$, the assumption $\lambda \leq 1$ becomes $\sigma^2 \leq \mu$. We see in Section 5.2 that this assumption is required if we want $P(a_1 \leq 0)$ to be negligible. Hence, in the following we always assume $\sigma^2 \leq \mu$, so that the function $\tilde{Z}$ is concave.

Because $\tilde{Z}$ is concave $\tilde{Z}$ has at most one maximum. Suppose that we can compute the maximum $y^*$ of $\tilde{Z}$ over $\mathbb{R}^+$, which could be greater than $\sum \mu_i$. The concavity of $\tilde{Z}$ implies that

$$y_1 \leq y_2 \leq y^* \Rightarrow \tilde{Z}(y_1) \leq \tilde{Z}(y_2) \quad \text{and} \quad y^* \leq y_2 \leq y_1 \Rightarrow \tilde{Z}(y_1) \leq \tilde{Z}(y_2),$$

for any $y_1, y_2 \in [1, \sum \mu_i]$. Recalling $y = \sum \mu_i x_i$, we can write a similar property for $Z_R$:

$$\sum \mu_i x_{1i} \leq \sum \mu_i x_{2i} \leq y^* \Rightarrow Z_R(x_1) \leq Z_R(x_2)$$

and

$$y^* \leq \sum \mu_i x_{2i} \leq \sum \mu_i x_{1i} \Rightarrow Z_R(x_1) \leq Z_R(x_2),$$
for any fractional vectors $x_1, x_2 \in \mathcal{B}$.

Hence, the closer $\sum \mu_i x_i$ is from $y^*$, the higher is $Z_R(x)$. More precisely, different cases might occur. If $\sum \mu_i \leq y^*$, the closest $y = \sum \mu_i x_i, x \in \mathcal{B}$, from $y^*$ is given by $x_i^* = 1$ for each $i \in N$. This $x^*$ is the solution to (17). If $\sum \mu_i > y^*$, we must look for $x_1^*$ and $x_2^*$ in $\mathcal{B}$ which minimize the distances between $\sum \mu_i x_i^*$ and $y^*$, where $\sum \mu_i x_i^* \leq y^*$ and $\sum \mu_i x_i^* \geq y^*$. Namely, we need first to solve two subset sum problems written below, where $y^*$ has been replaced by $\lfloor y^* \rfloor$ and $\lceil y^* \rceil$ because of the integrality of $\mu$ and $x$:

$$\max \sum_{i \in N} \mu_i x_i \quad \text{s.t.} \quad \sum_{i \in N} \mu_i x_i \leq \lfloor y^* \rfloor, \quad x_i \in \{0, 1\}, \quad (10)$$

and

$$\min \sum_{i \in N} \mu_i x_i \quad \text{s.t.} \quad \sum_{i \in N} \mu_i x_i \geq \lceil y^* \rceil, \quad x_i \in \{0, 1\}. \quad (11)$$

Denote by $x_1^*$ and $x_2^*$ solutions to (10) and (11). A solution to (6) is given by $x^* \in \mathcal{B}$ such that $Z_R(x^*) = \max(Z_R(x_1^*), Z_R(x_2^*))$.

Problems (10) and (11) are weakly polynomial, because they are particular cases of the knapsack problem. Furthermore, using their specific subset sum structure, there exists faster algorithms than for the general knapsack [10].

Finally, remark that $\lfloor y^* \rfloor$ and $\lceil y^* \rceil$ can be computed in $O(\log_2 \sum \mu_i)$, using a dichotomic search based on the sign of $Z'$.

2.3 Extensions

One reason which makes the deterministic knapsack so widely studied is that capacity constraints appear in many difficult combinatorial optimization problems. Hence it is natural to seek how to extend the framework from Section 2.1 to more complex stochastic models where uncertain parameters are described by Gaussian variables.

In this section, we present three direct extensions of the framework from Section 2.1 to other simple recourse problems. Given the good results obtained for the knapsack, see Section 5, we believe the following problems can be tackled efficiently by $NP/NLP$ algorithms.

2.3.1 Traveling salesman

A classical problem in the combinatorial optimization literature is the traveling salesman problem. Given a complete directed graph with nodes $N$, and travel costs $c_{ij}$, $i, j \in N$, the problem aims to find a cycle of minimum cost passing once through each node. A simple recourse stochastic extension considers a time limit $T$ and that each edge has a travel time, which is described by a random variable $a_{ij}$. If the total travel
time of a tour exceeds $T$, some penalty must be paid at the cost of $K$ per unit of delay. This results in the subsequent problem, which has been extended to a three-index model by Laporte et al. [17]:

$$\min \sum_{i,j \in N} c_{ij} x_{ij} + K \mathbb{E}_a \left[ \max \left( 0, \sum_{i,j \in N} a_{ij}(\omega)x_{ij} - T \right) \right] \quad (12)$$

subject to:

$$\sum_{j \in N} x_{ij} = 1 \quad i \in N$$

$$\sum_{i \in N} x_{ij} = 1 \quad j \in N$$

$$\sum_{i,j \in S} x_{ij} \leq |S - 1| \quad V \neq S \subset V$$

$$x_{ij} \in \{0, 1\}.$$ 

The only difference with the deterministic TSP is the presence of a non linear term in the objective (12), convex for most random variables. In particular, when $a_{ij} \sim \mathcal{N}(\mu_{ij}, \sigma^2_{ij})$, it can be simplified as in (4).

### 2.3.2 Multidimensional knapsack

Another extension of the knapsack considers $d$ capacity constraints instead of one:

$$\max_{x \in B} \left\{ \sum_{i \in N} p_i x_i \text{ s.t. } \sum_{j=1}^d a^j_i x_i \leq a^j_0, \ j = 1, \ldots, d \right\}. \quad (13)$$

This problem is also called "multidimensional knapsack". Its stochastic two-stages version is

$$\max_{x \in B} \left\{ \sum_{i \in N} p_i x_i - \sum_{j=1}^d K^j \mathbb{E}_{a_j} [y^j(\omega)] \text{ s.t. } \sum_{i \in N} a^j_i(\omega)x_i \leq a^j_0(\omega) + y^j(\omega), \ j = 1, \ldots, d \right\}. \quad (13)$$

As before, $y^j(\omega) = \max(0, \sum_{i \in N} a^j_i(\omega)x_i)$ so that (13) can be rewritten as

$$\max_{x \in B} \sum_{i \in N} p_i x_i - \sum_{j=1}^d K^j \mathbb{E}_{a_j} \left[ \max \left( 0, \sum_{i \in N} a^j_i(\omega)x_i - a^j_0(\omega) \right) \right].$$

Then, we can use the same procedure as before on each of the term $\mathbb{E}_{a_j} [\max(0, \sum_{i \in N} a^j_i(\omega)x_i - a^j_0(\omega))]$: if $a^j_i \sim \mathcal{N}(\mu^j_i, \sigma^2_i)$ for each $i \in N, j = 1, \ldots, d$ then $\mathbb{E}_{a_j} [\max(0, \sum_{i \in N} a^j_i(\omega)x_i - a^j_0(\omega))]$ can be replaced by $\mathbb{E}_{a_j} [\max(0, a^j(\omega)x_i)]$, where $a^j \sim \mathcal{N}(\mu^j(\omega), (\sigma^j(\omega))^2)$. Hence the objective is differentiable, concave and can be easily evaluated.

### 2.3.3 Multicommodity flow

The problem of computing a minimum cost multicommodity flow also presents many capacity constraints (14), sometimes called bundle constraints. Consider a directed
graph \((V, E)\), and a set of commodities \(h \in H\) with one origin \(o(h)\), one destination \(d(h)\) and demand value \(a^h\). The min cost flow can be formulated as:

\[
\min \sum_{e \in E, h \in H} p^h_ee^h \\
\text{s.t. } \sum_{h \in H} x^h_e = d^h \quad h \in H \\
\sum_{h \in H} x^h_e \leq c_e \quad e \in E \\
x \geq 0,
\] (14)

where \(N\) is the node-arc incidence matrix of directed graph \((V, E)\); \(d^h\) is a vector whose components corresponding to \(o(h)\) and \(d(h)\) are equal to \(a^h\) and \(-a^h\) respectively and other components are equal to 0; \(c_e\) is the capacity of arc \(e \in E\); \(x^h_e\) is the flow corresponding to commodity \(h \in H\) routed on arc \(e \in E\) and \(p^h_ee^h\) is its routing cost.

Next we consider the problem where demand values \(a^h\) are random variables. Then, there are many ways to derive a two-stage stochastic version of this problem, closely related to the stochastic network design problem. Most of them \([2, 18, 25]\) consider a recourse function where a penalty is paid for unmet demand. We consider next a model where the penalty is related to exceeded capacity:

\[
\min \sum_{e \in E, h \in H} p^h_ee^h + \sum_{e \in E} K_eE_e\left[y_e(\omega)\right] \\
\text{s.t. } \sum_{h \in H} a^h(\omega)x_e^h \leq c_e + y_e(\omega) \quad e \in E, \omega \in \Omega \\
0 \leq x \leq 1, \quad y \geq 0,
\] (15)

where \(1^h = d^h(\omega)/a^h(\omega)\) and \(x^h_e\) now represents the fraction of commodity \(h\) routed on arc \(e\). As before, we can get rid of the capacity constraints:

\[
\min \sum_{e \in E, h \in H} p^h_ee^h + \sum_{e \in E} K_eE_e \left[\max\left(0, \sum_{h \in H} a^h(\omega)x_e^h - c_e\right)\right] \\
\text{s.t. } N x^h = 1^h \quad h \in H \\
0 \leq x \leq 1.
\]

When the weights are independent Gaussian variables, \(a^h \sim N(\mu^h, (\sigma^h)^2)\), we can use results from Section 2 to rewrite the previous problem as

\[
\min \sum_{e \in E, h \in H} p^h_ee^h + \sum_{e \in E} K_e \left\{ \sigma_e(x)f\left(-\frac{\mu_e(x)}{\sigma_e(x)}\right) + \mu_e(x)G\left(-\frac{\mu_e(x)}{\sigma_e(x)}\right) \right\} \\
\text{s.t. } N x^h = 1^h \quad h \in H \\
0 \leq x \leq 1,
\] (16)

with \(\mu_e(x) = \sum_{h \in H} \mu^h_ee^h\) and \(\sigma_e(x) = \sqrt{\sum_{h \in H}(\sigma^h)^2(x^h_e)^2}\). Hence this problem turns out to be a convex multicommodity flow problems, which is a well studied problem \([22]\).
3 Constant weights

This section explores some properties of the problem when the only random variable is the capacity. First, we investigate the subset sum problem when the capacity follows any random variable, then we take a look at the general problem when the capacity is uniformly distributed.

3.1 Stochastic subset sum

We show next a result similar to Theorem 2 when $a_i, i \in N$, are given constants and $a_0$ is random variable satisfying one of the following assumptions:

(A1) $a_0$ has finite second moments

(A2) $a_0$ has a differentiable density probability function $f$.

Theorem 3 Consider the problem

$$\max_{x \in B} Z(x) = \max_{x \in B} \sum_{i \in N} a_i x_i - K\mathbb{E}_{a_0}[\max(0, \sum a_i x_i - a_0(\omega))], \quad (17)$$

where $a_0$ is a random variable. If $a_0$ satisfy (A1) or (A2), then (17) can be solved by a pseudo-polynomial algorithm in $O(n \sum a_i)$.

Proof The reduction from Proposition 1 holds when the variance of $a_0$ is zero and $K$ big enough. Let $Z$ be the objective function from (17). We must prove that $Z$ has the same properties as $Z_R$ from Section 2.2 so that the same argument as in the proof of Proposition 2 can be applied:

- $Z$ depends only on $y = \sum a_i x_i$; $Z(x) = \hat{Z}(y)$.
- $\hat{Z} : [0, \sum a_i] \rightarrow \mathbb{R}$ is concave.
- We can compute $[y^*]$ and $[y^*]$ in $O(n \sum a_i)$.

It is straightforward to see that $Z$ depends only upon $y = \sum a_i x_i$, with $\hat{Z}(y) = y - \mathbb{E}_{a_0}[\max(0, y - a_0(\omega))]$.

Then, the concavity of $Z$ (and therefore $\hat{Z}$) follows from Theorem 1, under the assumption that $a_0$ has finite second moments.

Otherwise, let us denote by by $F$ and $f$ the cumulative distribution and the density distribution of the random variable $a_0 : \Omega \rightarrow \mathbb{R}^+.$

We can rewrite the objective function:

$$Z(x) = \sum_{i \in N} a_i x_i - K \int_{\mathbb{R}_+} f(a_0) \max \left(0, \sum a_i x_i - a_0 \right) da_0$$

$$= \sum_{i \in N} a_i x_i - K \int_0^{\sum a_i x_i} f(a_0) \left(\sum a_i x_i - a_0 \right) da_0. \quad (18)$$

If $f$ is differentiable, recalling the following calculus formula,

$$\frac{d}{dy} \int_0^{h(y)} f(y, a_0) da_0 = \int_0^{h(y)} \frac{d}{dy} f(y, a_0) da_0 + f(y, h(y)) \frac{d}{dy} h(y),$$

we see that the second derivative of $\hat{Z}$ is always negative or zero:

$$\hat{Z}''(y) = -K f(y) \leq 0 \quad \forall y \geq 0.$$
Finally we can use a dichotomic search based on the sign of $\hat{Z}(x) - \hat{Z}(x + \delta)$, for $\delta$ small enough. If $f$ is differentiable, so is $\hat{Z}$ and this results in using again the sign of $\hat{Z}'$.

\[\square\]

3.2 General problem when $a_0$ is uniformly distributed

In this section, we consider that the profit vector $p$ is different from the weight vector $a$, and that all parameters are integers. In particular $K$ must be integer, whereas assuming $K \in \mathbb{R}_+$ was enough to prove the results from previous sections. Hence, we face again a concave optimization problem, that can be tackled using the algorithm developed in Section 4:

\[
\max_{x \in B} \sum_{i \in N} p_i x_i - K \int_0^{\sum a_i x_i} f(a_0) \left( \sum a_i x_i - a_0 \right) \, da_0. \tag{19}
\]

Now, assuming that $a_0$ is uniformly distributed between positive integers $a_0$ and $\bar{a}_0$ we show next that, under some assumption, the optimization problem is weakly NP-complete. Note that this problem can also be seen as a robust knapsack problem with linear penalty [21]. With $a_0$ uniformly distributed, the function to be maximized in (19) becomes:

\[
Z(x) = \sum_{i \in N} p_i x_i - \frac{K}{\bar{a}_0 - a_0} \int_{\min(\sum a_i x_i, \bar{a}_0)}^{\min(\sum a_i x_i, a_0)} \left( \sum a_i x_i - a_0 \right) \, da_0, \tag{20}
\]

where we assume $\sum a_i > a_0$ to avoid the trivial solution $x = (1, \ldots, 1)$. The rest of the section investigates the complexity of (20). Applying an argument similar to Proposition 1 to a problem with $\bar{a} = a + 1$, we have that it is in general NP-hard to maximize (20). Then decompose $Z$ into three functions:

\[
Z(x) = \begin{cases} 
Z_1(x) = \sum p_i x_i - \frac{K}{\bar{a}_0 - a_0} \left( \sum a_i x_i - a_0 \right)^2 & \text{for } \sum a_i x_i \leq a_0 \\
Z_2(x) = K \frac{\bar{a}_0 a_0 - a_0^2}{\bar{a}_0 - a_0} + \sum (p_i - Ka_i) x_i & \text{for } \bar{a}_0 \leq \sum a_i x_i \\
Z_3(x) = K \frac{\bar{a}_0 a_0 - a_0^2}{\bar{a}_0 - a_0} + \sum (p_i - Ka_i) x_i & \text{for } \bar{a}_0 \leq \sum a_i x_i
\end{cases}
\tag{21}
\]

Let $x^*, x_1^*, x_2^*$ and $x_3^*$ be optimal solutions of $\max_B Z(x)$, $\max_B \{Z_1(x) \text{ s.t. } \sum a_i x_i \leq a_0\}$, $\max_B Z_2(x)$ and $\max_B \{Z_3(x) \text{ s.t. } \sum a_i x_i \geq \bar{a}_0\}$ respectively. Then $Z(x^*) = \max(Z_1(x_1^*), Z_2(x_2^*), Z_3(x_2^*)$).

\[\text{Proof}\] The results follows from $Z_1(x) \geq Z_2(x)$ and $Z_3(x) \geq Z_2(x)$ for any $x \in B$. \[\square\]

Next we study the complexity of those three optimization problems. From Lemmas 4, 5, 6 and 7 we have:

**Theorem 4** Maximizing $Z$ over $B$ is in general NP-complete. However, if $\left( p_i + \frac{Ka_i}{\bar{a}_0 - a_0} (2a_0 - a_i) \right) \geq 0$, for each $i \in N$, $Z$ can be maximized in $O(nK \sum a_i)$.

For large number of items with individual volumes small enough, this condition is likely to be satisfied.

**Lemma 5** Maximizing $Z_1(x)$, for $x$ binary and $\sum a_i x_i \leq a_0$, can be done in $O(n \sum a_i)$.
Proof} This is a knapsack problem, which can be optimized in \( O(n a_0) \) and therefore in \( O(n) \) because \( a_0 < \sum a_i \).

\( \square \)

**Lemma 6** Maximizing \( Z_3(x) \), for \( x \) binary and \( \sum a_i x_i \geq \pi_0 \), can be done in \( O(n) \).

\textbf{Proof} In the following, we assume \( \sum a_i > \pi_0 \), otherwise \( \sum a_i x_i \) is always smaller than \( \pi_0 \) so that the problem does not have a solution. We show that this problem reduces to a small knapsack.

1. Define \( M = \{ i \in N \mid p_i - Ka_i < 0 \} \) and \( \tilde{a}_0 = \sum a_i - \pi_0 \); let \( x^* \) be the solution to the following knapsack \((x^*_i = 0 \text{ for } i \in N/M)\):

   \[
   \max_B \left( \sum_{i \in M} (Ka_i - p_i)x_i \text{ s.t. } \sum a_i x_i \leq \tilde{a}_0 \right).
   \]

2. The solution to our problem is \( x_i = 1 - x^*_i \), for each \( i \in N \).

Then because \( \tilde{a}_0 \leq \sum a_i \), the previous knapsack can be optimized in \( O(n) \).

\( \square \)

**Lemma 7** If \( \left(p_i + \frac{Ka_i}{2(a_0 - a_i)}(2a_0 - a_i)\right) \geq 0 \text{ for each } i \in N\), maximizing \( Z_2(x) \), for \( x \) binary, can be done in \( O(nK \sum a_i) \).

\textbf{Proof} Expanding the square in the objective of \( Z_2 \), and using that \( x^2_i = x_i \) because \( x \in B \), we get:

\[
- \frac{K a_0^2}{2(a_0 - a_i)} + \max_{x \in B} \left( \sum_{i \in N} \left(p_i + \frac{Ka_i}{2(a_0 - a_i)}(2a_0 - a_i)\right)x_i - \frac{K}{a_0 - a_i} \sum_{i, j \in N, i \neq j} a_i a_j x_i x_j \right).
\]

Assuming that \( \left(p_i + \frac{Ka_i}{2(a_0 - a_i)}(2a_0 - a_i)\right) \geq 0 \text{ for all } i \in N\), all coefficient are integer and \( (22) \) becomes a particular case of half-products [3]

\[
f = \sum_{i \in N} c_i x_i - \sum_{i, j \in N, i \neq j} a_i a_j x_i x_j,
\]

where \( c \mapsto \left(p + \frac{Ka}{2(a - a_i)}(2a - a_i)\right) \) and \( a = b \mapsto 2Ka \). Badics and Boros [3] proved that maximizing those functions for binary values is NP-complete, even when \( a = b \).

However, it is not strongly NP-complete, given a dynamic programming algorithm by Boros and Hammer [5] running in \( O(nA) \) for general half-products with positive coefficients, where \( A = 2K \sum a_i \).

\( \square \)

Besides this dynamic programming approach, new versions of optimization softwares, including CPLEX 11, can manage maximization of integer problems with a concave and quadratic objective function. Nevertheless, we show in Section 5 that the LP/NLP algorithm described in the next section solves \((22)\) much faster than does CPLEX 11.
4 LP/NLP Algorithm

In this Section, we briefly present an algorithm for solving the following kind of problem, also called MINLP (mixed integer non linear programming):

\[
\begin{align*}
\text{max} & \quad h(x) \\
\text{s.t.} & \quad Ax \leq b \\
& \quad x \in \mathbb{Z}_+^n,
\end{align*}
\]  

(23)

where \( h \) is assumed concave and differentiable. For more information about MINLP, we refer to the review by Grossman [8].

The algorithm is based on the work of Quesada and Grossman [23]. Our implementation is similar to FilMINT by Abysheck et al. [1], with the main difference that we use the branch-and-cut framework of CPLEX [9] whereas they use MINTO [24]. For the sake of simplicity we assume that all constraints are linear. Nevertheless, an extension of the subsequent algorithm can deal efficiently with convex and differentiable constraints as well as with real variables.

Because \( h \) is concave and differentiable, problem (23) is equivalent to

\[
\begin{align*}
\text{max} & \quad \theta \\
\text{s.t.} & \quad h(x^*) + \sum_{i=1}^{n} \frac{\partial h}{\partial x_i}(x^*)_i(x_i - x^*_i) \geq \theta, \quad x^* \in \mathbb{R}_+^n \\
& \quad Ax \leq b \\
& \quad x \in \mathbb{Z}_+^n,
\end{align*}
\]

which has an infinite number of constraints. The main idea of outer-approximation is that, for a given sensibility parameter \( \epsilon > 0 \), only a finite number of those constraints are required in a solution. For a given cut pool \( P \), we define the upper bounding problem

\[
\begin{align*}
\text{max} & \quad \theta \\
\text{s.t.} & \quad h(x^*) + \sum_{i=1}^{n} \frac{\partial h}{\partial x_i}(x^*)_i(x_i - x^*_i) \geq \theta, \quad x^* \in P \\
& \quad Ax \leq b \\
& \quad x \in \mathbb{Z}_+^n.
\end{align*}
\]

(24)

Our NP/NLP algorithm works in two steps. First, initialize \( P \) with Algorithm 1. Then solve (24) with the branch-and-cut described in Algorithm 2. \( T \) represents the branch-and-bound tree and solving a node \( o' \in T \) means solving the LP relaxation of (24) augmented with branching constraints of \( o' \).

5 Computational Results

In this section, we present computational results obtained using Algorithm 2 on two classes of problems:

- Weights are fixed and the capacity follows a uniform random variable, see Section 3.2.
Algorithm 1 Initializing $P$

Initialize cut pool $P$ to bound the LP, using for instance box constraints.

repeat

Solve the LP relaxation of (24). Let $(\theta^*, x^*)$ be an optimal solution.

if $\theta^* \geq h(x^*) + \epsilon$ then

Add the following cut to $P$

$$h(x^*) + \sum \frac{\partial h}{\partial x_i}(x^*) (x_i - x^*_i) \geq \theta.$$ 

end if

until $\theta^* < h(x^*) + \epsilon$

return $P$

Algorithm 2 LP/NLP algorithm

Require: A starting cut pool $P$.

Initialize the tree: $T = \{o\}$ where $o$ has no branching constraints; $\overline{\theta} = +\infty$.

while $T$ is nonempty do

Select a node $o' \in T$.

$T := T \setminus \{o'\}$

Solve $o'$. Let $(\theta^*, x^*)$ be an optimal solution.

if $\theta^* < \overline{\theta}$ then

if $x^*$ is fractional then

Branch, resulting in nodes $o^*$ and $o^{**}$, $T := T \cup \{o^*, o^{**}\}$.

else if $\theta^* \geq h(x^*) + \epsilon$ then

Add the following cut to $P$

$$h(x^*) + \sum \frac{\partial h}{\partial x_i}(x^*) (x_i - x^*_i) \geq \theta.$$ 

$T := T \cup \{o'\}$

else

Define a new upper bound $\overline{\theta} := \theta^*$ and save current solution, $\overline{x} = x^*$.

end if

end if

end if

end while

return $\overline{x}$

--- Weights follow Gaussian random variables independently distributed and the capacity is fixed, see Section 2.2.

Algorithms 1 and 2 are implemented within CPLEX, with $\epsilon = 0.1$. We use Cut-Callback, IncumbentCallback and BranchCallback to implement the different steps of Algorithm 2, which prevents from using the dynamic search. Then we keep default parameters concerning node selection, branching rules and generation of cuts, unless when a cut is added to $P$: then the cut is added as a global cut and as a branching constraint. All algorithms are coded in JAVA on a HP Compaq 6510b with a processor Intel Core 2 Duo of 2.40 GHz and 2 GB of RAM memory. The solution times reported are CPU times of the sum of the durations of both Algorithms.
5.1 Quadratic Instances

The aim of this Section is to compare Algorithm 2 and the general purpose mixed integer quadratic convex function solver of CPLEX 11 on problem (22). Note that we do not consider the small knapsacks in the subproblems (see Lemma 4) but only the solution of $Z_2$ unconstrained. Anyway, solution times required by the subproblems are order of magnitude smaller than the time needed to solve quadratic problem.

Table 1 Comparison between LP/NLP and CPLEX when $K$ increases, $n = 100$.

<table>
<thead>
<tr>
<th>K/R</th>
<th>10^3</th>
<th>10^4</th>
<th>10^3</th>
<th>10^4</th>
<th>10^3</th>
<th>10^4</th>
<th>Time ratios</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.019 ± 0.01</td>
<td>0.02 ± 0.01</td>
<td>0</td>
<td>0</td>
<td>0.019 ± 0.01</td>
<td>0.038 ± 1.04</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0.018 ± 0.01</td>
<td>0.02 ± 0.01</td>
<td>0</td>
<td>0</td>
<td>0.036 ± 0.04</td>
<td>0.943 ± 1.76</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0.02 ± 0.01</td>
<td>0.021 ± 0.01</td>
<td>0</td>
<td>0</td>
<td>0.246 ± 0.44</td>
<td>5.718 ± 10.5</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0.019 ± 0.01</td>
<td>0.023 ± 0.01</td>
<td>0</td>
<td>0</td>
<td>1.311 ± 2.5</td>
<td>9.913 ± 18.33</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0.02 ± 0.01</td>
<td>0.022 ± 0.01</td>
<td>0</td>
<td>0</td>
<td>2.598 ± 4.94</td>
<td>11.894 ± 21.29</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>0.02 ± 0.01</td>
<td>0.022 ± 0.01</td>
<td>0</td>
<td>0</td>
<td>3.233 ± 6.13</td>
<td>14.764 ± 25.03</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>0.021 ± 0.01</td>
<td>0.022 ± 0.01</td>
<td>0</td>
<td>0</td>
<td>3.299 ± 6.19</td>
<td>17.98 ± 28.62</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>0.021 ± 0.01</td>
<td>0.023 ± 0.01</td>
<td>0</td>
<td>0</td>
<td>3.329 ± 7.1</td>
<td>21.403 ± 32.12</td>
<td>1</td>
</tr>
<tr>
<td>18</td>
<td>0.022 ± 0.01</td>
<td>0.025 ± 0.01</td>
<td>0</td>
<td>0</td>
<td>4.662 ± 8.58</td>
<td>25.744 ± 38.1</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>0.021 ± 0.01</td>
<td>0.024 ± 0.01</td>
<td>0</td>
<td>0</td>
<td>5.256 ± 9.62</td>
<td>28.096 ± 40.2</td>
<td>0</td>
</tr>
</tbody>
</table>

We generated different sets of random instances as follows. We consider two data ranges: $R = 1000$ and $R = 10000$. Then the parameters $a_i$ and $p_i$ for each item $i \in \{1, \ldots, n\}$ are integers uniformly generated between 1 and $R$. For each data range $R$, value of the penalty factor $K$ and number of items $n$, we generate 100 instances, with $\mathbb{E}(a_0) = (h/101) \sum a_i$ for instance number $h$. The capacity varies uniformly between $90\% \mathbb{E}(a_0)$ and $110\% \mathbb{E}(a_0)$. We fix a time limit of 100 seconds per instance. We report on Tables 1 and 2 the average solution time in seconds and the dispersion for each group of 100 instances, and the number of unsolved instance for both algorithms. Note that the solution time has been set to 100 seconds for instances who could not be solved within 100 seconds. Hence, when there are unsolved instances in a group, the average, the standard deviation and the ratio for the that group are estimations. Ratios, given by $(\text{Time CPLEX}) / (\text{Time NP/NLP})$, are computed for each instance separately; we report the geometric average of those ratios, whereas we report the arithmetic average of solution times. We compute geometric average for ratios because of the following observation: if the ratio for one instance is 1/2 and the one for another instance is 2, their "average" should be equal to one, which is the case using the geometric average.

From Table 1, we see that the NP/NLP performance does not depend on the value of $K$, and that instances for the two range values are of the same difficulty. However CPLEX requires more time to solve instances with $R = 10^4$ than those with $R = 10^3$, this become even more significant when $K$ increases. Even tough CPLEX takes on average much more time than NP/NLP, the ratios close to one tell us that some instances are still solved much faster by CPLEX than by NP/NLP. In fact, CPLEX solution times vary a lot, from really small times to more than 100 seconds. In opposition, NP/NLP has always very small standard deviation.
Table 2 Comparison between LP/NLP and CPLEX when \( n \) increases, \( K = 2 \).

<table>
<thead>
<tr>
<th>( n/R )</th>
<th>( \text{NP/NLP} )</th>
<th>( \text{Time} )</th>
<th>( 10^3 )</th>
<th>( 10^4 )</th>
<th>( \text{Unsolved} )</th>
<th>( 10^3 )</th>
<th>( 10^4 )</th>
<th>( \text{Time} )</th>
<th>( 10^3 )</th>
<th>( 10^4 )</th>
<th>( \text{Unsolved} )</th>
<th>( 10^3 )</th>
<th>( 10^4 )</th>
<th>( \text{Time ratios} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>0.029 ± 0.01</td>
<td>( 0.052 ± 0.01 )</td>
<td>( 0.08 ± 0.05 )</td>
<td>( 0.269 ± 0.43 )</td>
<td>0</td>
<td>0</td>
<td>1.55</td>
<td>1.77</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>400</td>
<td>0.052 ± 0.01</td>
<td>( 0.057 ± 0.01 )</td>
<td>( 0.484 ± 0.47 )</td>
<td>( 0.239 ± 0.03 )</td>
<td>0</td>
<td>0</td>
<td>5.33</td>
<td>4.14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>600</td>
<td>0.076 ± 0.01</td>
<td>( 0.081 ± 0.01 )</td>
<td>( 1.878 ± 3.4 )</td>
<td>( 0.638 ± 0.07 )</td>
<td>1</td>
<td>0</td>
<td>9.51</td>
<td>7.75</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>800</td>
<td>0.097 ± 0.01</td>
<td>( 0.109 ± 0.01 )</td>
<td>( 2.083 ± 1.45 )</td>
<td>( 1.447 ± 0.15 )</td>
<td>0</td>
<td>0</td>
<td>14.83</td>
<td>13.31</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>0.12 ± 0.02</td>
<td>( 0.129 ± 0.02 )</td>
<td>( 3.814 ± 2.65 )</td>
<td>( 2.607 ± 0.16 )</td>
<td>0</td>
<td>0</td>
<td>22.19</td>
<td>20.26</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1200</td>
<td>0.144 ± 0.02</td>
<td>( 0.166 ± 0.03 )</td>
<td>( 4.176 ± 3.0 )</td>
<td>( 4.064 ± 1.09 )</td>
<td>2</td>
<td>1</td>
<td>28.52</td>
<td>28.29</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1400</td>
<td>0.174 ± 0.02</td>
<td>( 0.19 ± 0.03 )</td>
<td>( 5.797 ± 2.17 )</td>
<td>( 5.727 ± 0.36 )</td>
<td>1</td>
<td>0</td>
<td>31.88</td>
<td>30.33</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1600</td>
<td>0.203 ± 0.02</td>
<td>( 0.222 ± 0.03 )</td>
<td>( 6.982 ± 2.08 )</td>
<td>( 7.984 ± 0.5 )</td>
<td>1</td>
<td>0</td>
<td>34.8</td>
<td>36.09</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1800</td>
<td>0.225 ± 0.03</td>
<td>( 0.249 ± 0.03 )</td>
<td>( 10.708 ± 3.01 )</td>
<td>( 11.142 ± 0.6 )</td>
<td>0</td>
<td>0</td>
<td>43.05</td>
<td>44.94</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2000</td>
<td>0.264 ± 0.03</td>
<td>( 0.281 ± 0.03 )</td>
<td>( 12.313 ± 3.64 )</td>
<td>( 14.309 ± 1.18 )</td>
<td>2</td>
<td>0</td>
<td>48.28</td>
<td>50.86</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2 studies the impact of increasing the number \( n \) of items, hence variables in the model. It is clear from the ratios values that the NP/NLP handle much better bigger instances than CPLEX does. Indeed the ratio average increases more or less linearly with the number of items. Recalling that CPLEX deals with \( O(n^2) \) variables it is natural that the solution time is very impacted by \( n \). On the other hand, the number of variables in NP/NLP only increases linearly with \( n \), because this algorithm deals implicitly with the non linear objective. However, because \( K \) is small enough, CPLEX like NP/NLP solves instances with \( R = 10^3 \) as well as instances with \( R = 10^4 \). Note that even though CPLEX is outperformed by LP/NLP, the solution times do not grow too fast with the problems size. We stopped our test to 2000 variables because CPLEX required almost all memory to solve those instances.

5.2 Gaussian Instances

In this section, we consider the more complex objective (5). Note that function \( G \) is read from a table, avoiding the burden of computing the integral. The instances are generated as in the quadratic case, with the subsequent modifications: parameters \( \mu_i \) and \( p_i \) for each item \( i \in N \) are integers uniformly generated between 1 and \( R \). Each variance \( \sigma_i \) is an integer uniformly generated between 1 and \( \lfloor \mu/4 \rfloor \) for each item \( i \in N \), to avoid negative outcomes as explained next. As before, we generated 100 instances for each value of parameters \( K \), \( n \) and \( R \), with constant capacity \( a_0 = (h/101) \sum \mu_i \) for instance number \( h \); all results take the average over those groups of 100 instances. We do not report the standard deviations anymore, as they stay as small as for the quadratic problems, increasing slowly with the number of variables.

Note that Gaussian variables can take negative values, so that for some scenarios, some weights are negative, which does not make sense. However, when the ratio \( \sigma/\mu \) is small enough, those scenarios are negligible. For instance, the probability that \( \mathcal{N}(\mu, \sigma^2) \) takes a value less than \( \mu - 4\sigma \) is a bit less than 0.0001. Therefore we will always assume \( \mu/4\sigma \geq 1 \) when generating our instances.

The following results show that the resolution time behave as in the quadratic case, regarding the value of the penalty factor and the number of items. Then, we study more specifically the time spent in Algorithm 1 to generate cut pool \( P \). The subsequent tables report the total time in seconds (Time), the fraction of time spent
to generate $P$ (Initialization), the number of cuts generated initially and the number of cuts added in the course of Algorithm 2.

### Table 3  Uncorrelated Instances, $n = 500$.  

<table>
<thead>
<tr>
<th>K/R</th>
<th>Time Initialization</th>
<th>InitCuts $10^3$</th>
<th>InitCuts $10^4$</th>
<th>AddCuts $10^3$</th>
<th>AddCuts $10^4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.346 0.332</td>
<td>65% 65%</td>
<td>8.33 8.26</td>
<td>0.69 0.73</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.369 0.357</td>
<td>64% 64%</td>
<td>8.83 8.73</td>
<td>0.97 0.85</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0.358 0.357</td>
<td>64% 65%</td>
<td>8.98 8.98</td>
<td>0.98 0.78</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0.362 0.36</td>
<td>65% 66%</td>
<td>9.09 9.06</td>
<td>0.79 0.78</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0.343 0.341</td>
<td>64% 63%</td>
<td>9.15 9.18</td>
<td>0.8 0.91</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>0.341 0.341</td>
<td>63% 62%</td>
<td>9.23 9.19</td>
<td>0.98 1.0</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>0.341 0.379</td>
<td>65% 65%</td>
<td>9.33 9.28</td>
<td>0.79 0.84</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>0.352 0.363</td>
<td>67% 66%</td>
<td>9.37 9.1</td>
<td>0.7 0.78</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>0.381 0.361</td>
<td>66% 64%</td>
<td>9.46 9.14</td>
<td>0.76 0.86</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>0.386 0.369</td>
<td>65% 66%</td>
<td>9.71 9.3</td>
<td>0.88 0.81</td>
<td></td>
</tr>
</tbody>
</table>

We can see on Table 3 that the penalty factor $K$ has little influence on the $LP/NLP$ algorithm, apart from the case $K = 2$ which seems easier. Therefore, in the subsequent instances, we fix $K = 10$. Results from Table 4 show that we can easily solve problems up to 5000 variables, even though times are significantly bigger than in the deterministic case. For example, uncorrelated instances with 5000 items are solved by Martello et al. [19] on average in 0.01 seconds, whereas we need on average 19 seconds to solve problems with 5000 items. Note that an important fraction of the time is spent in the generation of the cut pool, and that that fraction increases with the problem size.

### Table 4  Uncorrelated Instances, $K = 10$.  

<table>
<thead>
<tr>
<th>n/R</th>
<th>Time $10^3$</th>
<th>Time $10^4$</th>
<th>Initialization $10^3$</th>
<th>Initialization $10^4$</th>
<th>InitCuts $10^3$</th>
<th>InitCuts $10^4$</th>
<th>AddCuts $10^3$</th>
<th>AddCuts $10^4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>0.072</td>
<td>0.073</td>
<td>54% 49%</td>
<td>9.04 8.77</td>
<td>1.09 1.28</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>400</td>
<td>0.209</td>
<td>0.193</td>
<td>65% 62%</td>
<td>9.08 8.75</td>
<td>0.98 0.95</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>600</td>
<td>0.381</td>
<td>0.353</td>
<td>70% 72%</td>
<td>8.85 9.03</td>
<td>0.94 0.59</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>800</td>
<td>0.616</td>
<td>0.579</td>
<td>75% 77%</td>
<td>8.96 8.97</td>
<td>0.66 0.49</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>0.896</td>
<td>0.876</td>
<td>78% 78%</td>
<td>8.73 8.92</td>
<td>0.61 0.61</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2000</td>
<td>3.447</td>
<td>3.361</td>
<td>83% 84%</td>
<td>8.81 9.2</td>
<td>0.5 0.42</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3000</td>
<td>7.329</td>
<td>7.5</td>
<td>86% 87%</td>
<td>8.85 9.22</td>
<td>0.34 0.28</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4000</td>
<td>13.302</td>
<td>12.682</td>
<td>86% 87%</td>
<td>8.93 8.9</td>
<td>0.42 0.25</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5000</td>
<td>20.405</td>
<td>17.462</td>
<td>86% 86%</td>
<td>9.02 7.8</td>
<td>0.4 0.28</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Pursuing our comparison with the deterministic knapsack, we wondered whether strongly correlated instances are harder to solve than uncorrelated ones. Recall that strongly correlated instances are characterized by the relations $p_i = \mu_i + R/10$, $i \in N$. Results from Table 5 show that those instances are roughly of the same difficulty as the uncorrelated ones, whereas the difference is very significant in the deterministic case. Hence, it seems that replacing the capacity constraint by a penalty term in the objective provides the problem with a structure really different from the deterministic case. Correlated instances for the stochastic problem require less cutting planes in the
initial phase but more in the branch-and-cut phase than uncorrelated ones, especially for smaller instances.

Table 5 Strongly Correlated Instances, $K = 10$.

<table>
<thead>
<tr>
<th>n/R</th>
<th>Time</th>
<th>Initialization</th>
<th>InitCuts</th>
<th>AddCuts</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$10^3$</td>
<td>$10^4$</td>
<td>$10^3$</td>
<td>$10^4$</td>
</tr>
<tr>
<td>200</td>
<td>0.122</td>
<td>0.139</td>
<td>29%</td>
<td>26%</td>
</tr>
<tr>
<td>400</td>
<td>0.353</td>
<td>0.374</td>
<td>33%</td>
<td>31%</td>
</tr>
<tr>
<td>600</td>
<td>0.927</td>
<td>0.718</td>
<td>25%</td>
<td>33%</td>
</tr>
<tr>
<td>800</td>
<td>0.842</td>
<td>1.064</td>
<td>47%</td>
<td>36%</td>
</tr>
<tr>
<td>1000</td>
<td>2.09</td>
<td>1.006</td>
<td>30%</td>
<td>57%</td>
</tr>
<tr>
<td>2000</td>
<td>3.155</td>
<td>2.727</td>
<td>76%</td>
<td>78%</td>
</tr>
</tbody>
</table>

Let us see the practical interest of Theorem 2 on subset sum instances. We generated instances as in the Gaussian case with the following differences: $\mu_i$ are uniformly generated between 4 and $R$ and $\sigma_i = \sqrt{\lambda \mu_i}$, where we choose $\lambda = 4$ to respect also the condition $\mu/4\sigma \geq 1$. We can see on Table 6 that the solution times obtained are order of magnitudes higher than those from Martello et al. Therefore, it is faster in practice to use the equivalence proved with Theorem 2 and to solve two deterministic subset sum problems instead of one stochastic problem. Now, comparing Tables 4 and 6 we see that the initial cut pool contains less cuts for the subset sum problem than for the general problem. This tendency is inverted for cuts generated during the branch-and-cut algorithm. This can be explained as follows: in the subset sum case, the linear relaxation of the objective depends only on one variable so that its structure is simpler than for the general problem. However, when dealing with the integrality constraints, those missing cuts are required as well as some additional ones to brake the symmetry among variables.

Table 6 Subset Sum, $\lambda = 4$ and $K = 10$.

<table>
<thead>
<tr>
<th>n/R</th>
<th>Time</th>
<th>Initialization</th>
<th>InitCuts</th>
<th>AddCuts</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$10^3$</td>
<td>$10^4$</td>
<td>$10^3$</td>
<td>$10^4$</td>
</tr>
<tr>
<td>200</td>
<td>0.101</td>
<td>0.122</td>
<td>31%</td>
<td>30%</td>
</tr>
<tr>
<td>400</td>
<td>0.227</td>
<td>0.284</td>
<td>50%</td>
<td>39%</td>
</tr>
<tr>
<td>600</td>
<td>0.449</td>
<td>0.51</td>
<td>55%</td>
<td>46%</td>
</tr>
<tr>
<td>800</td>
<td>0.639</td>
<td>0.801</td>
<td>61%</td>
<td>49%</td>
</tr>
<tr>
<td>1000</td>
<td>0.932</td>
<td>1.192</td>
<td>67%</td>
<td>54%</td>
</tr>
<tr>
<td>2000</td>
<td>3.097</td>
<td>3.919</td>
<td>74%</td>
<td>60%</td>
</tr>
</tbody>
</table>

Finally comparing Tables 6 and 7, the value of $\lambda = \mu_i/\sigma^2$ does not seem to influence the difficulty of the problems.

References

Table 7 Subset Sum, $\lambda = 10$ and $K = 10.$

<table>
<thead>
<tr>
<th>n/R</th>
<th>Time $10^3$</th>
<th>Time $10^4$</th>
<th>Initialization $10^3$</th>
<th>Initialization $10^4$</th>
<th>InitCuts $10^3$</th>
<th>InitCuts $10^4$</th>
<th>AddCuts $10^3$</th>
<th>AddCuts $10^4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>0.108</td>
<td>0.113</td>
<td>36%</td>
<td>31%</td>
<td>8.28</td>
<td>8.12</td>
<td>3.64</td>
<td>4.69</td>
</tr>
<tr>
<td>400</td>
<td>0.278</td>
<td>0.302</td>
<td>43%</td>
<td>38%</td>
<td>8.55</td>
<td>8.18</td>
<td>2.92</td>
<td>3.96</td>
</tr>
<tr>
<td>600</td>
<td>0.487</td>
<td>0.551</td>
<td>51%</td>
<td>42%</td>
<td>8.43</td>
<td>8.07</td>
<td>3.09</td>
<td>4.86</td>
</tr>
<tr>
<td>800</td>
<td>0.7</td>
<td>0.873</td>
<td>59%</td>
<td>47%</td>
<td>8.52</td>
<td>8.32</td>
<td>2.41</td>
<td>4.66</td>
</tr>
<tr>
<td>1000</td>
<td>1.04</td>
<td>1.208</td>
<td>61%</td>
<td>50%</td>
<td>8.45</td>
<td>8.16</td>
<td>2.31</td>
<td>4.58</td>
</tr>
<tr>
<td>2000</td>
<td>3.537</td>
<td>4.704</td>
<td>70%</td>
<td>49%</td>
<td>8.56</td>
<td>8.26</td>
<td>1.89</td>
<td>6.38</td>
</tr>
</tbody>
</table>


