Upper bounds for the 0-1 stochastic knapsack problem and a B&B algorithm*

Stefanie Kosuch

January 29, 2010

Abstract

In this paper we study and solve two different variants of static knapsack problems with random weights: The stochastic knapsack problem with simple recourse as well as the stochastic knapsack problem with probabilistic constraint. Special interest is given to the corresponding continuous problems and three different problem solving methods are presented. The resolution of the continuous problems allows to provide upper bounds in a branch-and-bound framework in order to solve the original problems. Numerical results on a dataset from the literature as well as a set of randomly generated instances are given.

1 Introduction

The *knapsack problem* has been widely studied for the last decades (Kellerer et al. (2004), Harvey M. Salkin (2006)). The problem consists in choosing a subset of items that maximizes an objective function w.r.t. a given capacity constraint. More precisely, we assume each item to have a benefit or benefit per weight unit as well as a specific weight or resource. Then, our aim is to choose a subset of items in order to maximize the total benefit w.r.t. a given capacity. There is a wide range of real life applications of the knapsack problem, amongst all transportation, finance, e.g. the purchase of commodities or stocks with a limited budged or schedule planning, where different tasks with different priority or benefit should be fulfilled in a limited time.

The knapsack problem is a combinatorial problem: each item is modeled by a binary decision variable $x \in \{0, 1\}$ with $x = 1$ if the item is chosen and 0 otherwise.

The knapsack problem is generally linear, i.e. both the objective function and the constraints are linear. Nevertheless, it is known to be NP-hard (see (Kellerer et al., 2004)).

In the deterministic case, all parameters (item weights, benefits and capacity) are known. However, in real life problems it is not uncommon that not all of the values are predetermined. These values can be modeled by continuously or discretely distributed random variables which turns the underlying problem into a *stochastic optimization problem* (for a

*Based on the article of the same title by S. Kosuch and A. Lisser accepted for publication in *Annals of Operations Research*

survey on optimization under uncertainty see (Sahinidis, 2004). As the deterministic problem, the stochastic knapsack problem is at least NP-hard (see (Kellerer et al., 2004)).
In this paper, the item weights are supposed to be independently normally distributed with known mean and variance, whilst the capacity and benefits remain deterministic. Readers interested in the case of random returns are referred to Henig (1990), Carraway et al. (1993) as well as to Morton and Wood (1997). In the latter, the authors solve this variant of a stochastic knapsack problem using dynamic and integer programming as well as a Monte Carlo solution procedure. In all three publications, the authors solve so called *stochastic target achievement problems*. This means that, instead of maximizing the expected reward, the objective of the problem is to maximize the probability to attain a certain target.
We consider two models of stochastic knapsack problems with random weights. The first is an unconstrained problem, namely the *Stochastic Knapsack Problem with simple recourse*, while the second is a constrained stochastic knapsack problem.
There are very few publications dealing with the (exact or approximated) solution of one of the problem types addressed in this paper.
A handful of publications dealing with the first formulation exists in the literature, see for instance Ağralıand Geunes (2008), Claro and de Sousa (2008), Cohn and Barnhart (1998) or Kleywegt et al. (2001). In (Cohn and Barnhart, 1998) and (Kleywegt et al., 2001) the authors also assume the weights to be normally distributed. In the latter, a Sample Average Approximation (SAA) Method is used to solve the problem approximatively. Our work is mainly inspired by the work of (Cohn and Barnhart, 1998) who used a branch-and-bound algorithm (hereafter called B&B algorithm) to solve the problem exactly. In Ağralıand Geunes (2008) the authors assume the item weights to follow a Poisson distribution. Like proceeded in this paper, they solve the continuous relaxation of their problem in order to compute upper bounds for a B&B algorithm. The authors of (Claro and de Sousa, 2008) solve the Stochastic Knapsack Problem with simple recourse in a very different way. As the problem can be seen as a multi-objective optimization problem, they solve i.a. Conditional Value at Risk (CVaR) reformulations using an SAA method as well as tabu search related techniques.
The second formulation of the stochastic knapsack problem addressed in this paper belongs to the general chance constrained stochastic optimization problems especially studied by Prekopa (1995) for continuous problems. In (Kleinberg et al., 1997) and (Goel and Indyk, 1999) the authors present approximation algorithms for three different combinatorial stochastic problems. One of these problems is a constrained stochastic knapsack problem with random weights. While in the former the item weights are assumed to be arbitrarily distributed which leads to an $O(1)$ approximation algorithm, the authors of the latter assume Poisson and exponentially distributed weights which allows them to prove the existence of a polynomial approximation scheme.
In this paper, we give special regard to the solution of the relaxed, i.e. continuous versions of the treated problems.
Two of these methods are *stochastic gradient type algorithms*. First papers on this iterative stochastic approximation methods where released in the middle of the last century (Robbins and Monro (1951), Kieper and Wolfowitz (1952)). Since then, an extensive amount of theoretical results on the convergence of the stochastic gradient algorithm and its variants has been published (Polyak (1990), L'Écuyer and Yin (1998)). The method has found many applications, particularly in machine learning and control theory. For a survey, see the books

by Nevel'son and Has'minskii (1976) and by Kushner and Yin (2003).
The third problem solving method presented is based on the reformulation of the stochastic problem as an equivalent deterministic problem, more precisely as a *Second Order Cone Programming* problem (see (Boyd et al., 1998)). This special type of convex optimization problem is most efficiently solved using interior point methods (Boyd and Vandenberghe, 2004).
The results obtained by studying the relaxed problem are afterwards used to provide upper bounds in a B&B algorithm. The B&B algorithm is one of the most common ways to solve deterministic knapsack problems. One of the first papers in which the author solved the knapsack problem using a B&B algorithm was (Kolesar, 1967). In (Martello and Toth, 1977) the authors present a method to calculate upper bounds for the $0-1$ knapsack problem and use them within a B&B algorithm. Recent work has been published in (Sun et al., 2007) where the authors present a B&B algorithm for the more general polynomial knapsack problem. In Carraway et al. (1993), the authors solve a variant of the stochastic knapsack problem using a B&B algorithm.
The problems studied in this paper are *static*, i.e. the decision which items to choose is made before the stochastic parameters come to be known. The majority of the papers on the stochastic knapsack problem studies the dynamic or "on-line" variant of the problem. In the case of the *dynamic stochastic knapsack problem*, the items (e.g. their reward and/or measures) are supposed to come to be known during the decision process either directly before or after an item has been chosen. Further decisions are therefore based on the weight parameters already revealed and the decision previously made. The problem consists therefor mostly in creating an optimal decision policy. For further reading see (Lin et al., 2008), (Babaioff et al., 2007), (Kleywegt and Papastavrou, 2001), (Marchetti-Spaccamela and Vercellis, 1995) or (Ross and Tsang, 1989).
Another important field of research concerning the stochastic knapsack problem are *approximation algorithms* as proposed by Kleinberg et al. (1997), Goel and Indyk (1999) or Klopfenstein and Nace (2006). In the latter, the authors use robust and dynamic programming to find feasible solutions for the chance constrained knapsack problem with random weights. In (Dean et al., 2004) the focus lies on the comparison of adaptive and non-adaptive policies for a stochastic knapsack problem where the size of each item is random but is revealed when the item is chosen. In a recent paper by Lu (2008), the author develops an approximation scheme in a rather uncommon way using differential equations and fluid and diffusion approximation approaches.
The remainder of this paper is organized as follows: In section 2, the mathematical formulations of the stochastic knapsack problems addressed in this paper are given. The different solving methods for the corresponding relaxed problems as well as the B&B algorithm are introduced in section 3. Numerical results are presented and discussed in section 4 and concluding remarks given in section 5.

2 Mathematical formulations

We consider a stochastic knapsack problem of the following form: Given a set of n items. Each item has a weight that is not known in advance, i.e. the decision of which items to choose has to be made without the exact knowledge of their weights. Therefore, we handle the weights as random variables and assume that weight χ_i of item i is independently normally distributed with mean $\mu_i > 0$ and standard deviation σ_i. Furthermore, each item has a predetermined reward per weight unit $r_i > 0$. The choice of a reward per weight unit can be justified by the fact that the value of an item often depends on its weight which we do not know in advance. We denote by χ, μ, σ and r the corresponding n-dimensional vectors. Our aim is to maximize the expected total reward $\mathbb{E}[\sum_{i=1}^n r_i \chi_i x_i]$. Our knapsack problem has a given weight capacity $c > 0$ but due to the stochastic nature of the weights the objective to respect this restriction can be interpreted in different ways. We consider two variants of stochastic knapsack problems. The second variant is studied in two equivalent formulations:

1. **The Stochastic Knapsack Problem with simple recourse** ($SRKP$)

$$\max_{x \in \{0,1\}^n} \mathbb{E}[\sum_{i=1}^n r_i \chi_i x_i] - d \cdot \mathbb{E}[[g(x,\chi) - c]^+] \tag{1}$$

2. **The Constrained Knapsack Problem** (CKP)

   a) **The Chance Constrained Knapsack Problem** ($CCKP$)

$$\max_{x \in \{0,1\}^n} \mathbb{E}[\sum_{i=1}^n r_i \chi_i x_i] \tag{2}$$

$$\text{s.t.} \quad \mathbb{P}\{g(x,\chi) \leq c\} \geq p \tag{3}$$

   a) **The Expectation Constrained Knapsack Problem** ($ECKP$)

$$\max_{x \in \{0,1\}^n} \mathbb{E}[\sum_{i=1}^n r_i \chi_i x_i] \tag{4}$$

$$\text{s.t.} \quad \mathbb{E}[\mathbb{1}_{\mathbb{R}^+}(c - g(x,\chi))] \geq p \tag{5}$$

where $\mathbb{P}\{A\}$ denotes the probability of an event A, $\mathbb{E}[\cdot]$ the expectation, $\mathbb{1}_{\mathbb{R}^+}$ the indicator function of the positive real interval, $g(x,\chi) := \sum_{i=1}^n \chi_i x_i$, $[x]^+ := \max(0,x) = x \cdot \mathbb{1}_{\mathbb{R}^+}(x)$ $(x \in \mathbb{R})$, $d \in \mathbb{R}^+$ and $p \in (0.5, 1]$ is the prescribed probability.
We call *solution vector* every $x \in \mathbb{R}^n$ such that $x = \arg\max_{x \in X_{ad}} J(x,\chi)$ where J is the objective function of one of the above maximization problems and $X_{ad} \subseteq \mathbb{R}^n$ the feasible set. We refer to the objective function maximum value of one of these problems as *solution value*.
Throughout this paper, we denote by f and F density and cumulative distribution function of the standard normal distribution, respectively.

3 Problem solving methods

This section is subdivided into two subsections: In the first one we present three possibilities the solve the relaxed stochastic knapsack problem, one for each formulation presented in section 2. In the second subsection, we use these methods to calculate upper bounds for a B&B algorithm in order to solve the corresponding combinatorial problems.

3.1 Calculating upper bounds

3.1.1 The stochastic knapsack problem with simple recourse

In this formulation, the capacity constraint has been included in the objective function by using the penalty function $[\cdot]^+$ and a penalty factor $d > 0$. This can be interpreted as follows: in the case where our choice of items leads to a capacity excess, a penalty occurs per overweight unit.

In order to simplify references to the included functions, we define

$$\psi_1(x,\chi) := \mathbb{E}\left[\sum_{i=1}^{n} r_i \chi_i x_i\right] \text{ and } \psi_2(x,\chi) := \mathbb{E}\left[[g(x,\chi)-c]^+\right]$$

i.e. our objective function becomes $J(x,\chi) = \psi_1(x,\chi) - d \cdot \psi_2(x,\chi)$.

We define a new random variable $X := g(x,\chi)$ which is normally distributed with mean $\hat{\mu} := \sum_{i=1}^{n} \mu_i x_i$, standard deviation $\hat{\sigma} := \sqrt{\sum_{i=1}^{n} \sigma_i^2 x_i^2}$, density function $\varphi(x) = \frac{1}{\hat{\sigma}} f(\frac{x-\hat{\mu}}{\hat{\sigma}})$ and cumulative distribution function $\Phi(x) = F(\frac{x-\hat{\mu}}{\hat{\sigma}})$. Based on these definitions, we can rewrite our objective function J in a deterministic way using the following:

$$\begin{aligned}
\mathbb{E}\left[[X-c]^+\right] &= \int_{-\infty}^{\infty} [X-c]^+ \cdot \varphi(X)\,\mathrm{d}X = \int_{c}^{\infty} (X-c)\cdot\varphi(X)\,\mathrm{d}X \\
&= \int_{c}^{\infty} X\cdot\varphi(X)\,\mathrm{d}X - c\int_{c}^{\infty} \varphi(X)\,\mathrm{d}X \\
&= \hat{\mu}\int_{c}^{\infty} \varphi(X)\,\mathrm{d}X + \hat{\sigma}^2\int_{c}^{\infty} \varphi'(X)\,\mathrm{d}X - c\int_{c}^{\infty} \varphi(X)\,\mathrm{d}X \\
&= \hat{\sigma}^2[\varphi(X)]_c^{\infty} + (\hat{\mu}-c)\,[\Phi(X)]_c^{\infty} = \hat{\sigma}^2\varphi(c) + (\hat{\mu}-c)\,[1-\Phi(c)] \\
&= \hat{\sigma}\cdot f(\frac{c-\hat{\mu}}{\hat{\sigma}}) + (\hat{\mu}-c)\cdot\left[1-F(\frac{c-\hat{\mu}}{\hat{\sigma}})\right]
\end{aligned}$$

This leads to the deterministic equivalent objective function

$$J_{det}(x) = \sum_i r_i \mu_i x_i - d\cdot\left[\hat{\sigma}\cdot f\left(\frac{c-\hat{\mu}}{\hat{\sigma}}\right) - (c-\hat{\mu})\cdot\left[1-F\left(\frac{c-\hat{\mu}}{\hat{\sigma}}\right)\right]\right] \tag{6}$$

The fact, that $SRKP$ admits a deterministic equivalent problem is important as we would like to solve the combinatorial problem using a B&B algorithm. This requires constant evaluations of the objective function.

As in the continuous case x_i is defined on the interval $[0,1]$, $SRKP$ becomes a concave optimization problem. Due to this concavity and as the objective function handles the capacity constraint, we can apply a stochastic gradient algorithm (see Algorithm 3.1).

A stochastic gradient algorithm is an algorithm that combines both Monte-Carlo techniques and the gradient method often used in optimization theory. Here, the former is used to approximate the gradient of the objective function that is a function in expectation. More precisely, if the objective function is $J(x,\chi) = \mathbb{E}[j(x,\chi)]$, we use in step $k+1$ the gradient $\nabla_x j(x^k, \chi^k)$ (where χ^k is a random sample of χ) instead of $\nabla_x J(x^k, \chi)$.

<u>Stochastic Gradient Algorithm</u>

- Choose x^0 in $X_{ad} = [0,1]^n$
- At step $k+1$, draw a sample $\chi^k = (\chi_1^k, ..., \chi_n^k)$ of χ according to its normal distribution
- Update x^k as follows:

$$x^{k+1} = x^k + \epsilon^k r^k$$

  where $r^k = \nabla_x j(x^k, \chi^k)$ and $(\epsilon^k)_{k\in\mathbb{N}}$ is a σ-sequence
- For all $i = 1, ..., n$: If $x_i^{k+1} > 1$ set $x_i^{k+1} = 1$ and if $x_i^{k+1} < 0$ set $x_i^{k+1} = 0$

Algorithm 3.1:

In the case of $SRKP$, we have $j(x,\chi) = \sum_i r_i \chi_i x_i - d \cdot [g(x,\chi) - c]^+$. As j is not differentiable, we approximate its gradient by using approximation by convolution (for further details on this method see (Ermoliev et al., 1995) or (Andrieu et al., 2007)). The basic idea of this method, which we simply call "approximation by convolution (method)", is to approximate the indicator function $\mathbb{1}_{\mathbb{R}^+}$ by its convolution with a function $h_t(x) := \frac{1}{t} h\left(\frac{x}{t}\right)$ that approximates the Dirac function when the parameter t goes to zero. The convolution of two functions is defined as follows:

$$(\rho * h)(x) := \int_{-\infty}^{\infty} \rho(y) h(x-y) \, \mathrm{d}y$$

Using a pair, continuous and non-negative function h with $\int_{-\infty}^{\infty} h(x) \, \mathrm{d}x = 1$ having its maximum in 0, we get the following approximation of a locally integrable real valued function ρ:

$$\rho_t(x) := (\rho * h_t)(x) = \frac{1}{t} \int_{-\infty}^{\infty} \rho(y) h\left(\frac{y-x}{t}\right) \, \mathrm{d}y$$

In the case of $\rho = \mathbb{1}_{\mathbb{R}^+}$, we have:

$$\rho_t(x) = \frac{1}{t}\int_0^{\infty} h\left(\frac{y-x}{t}\right) \,\mathrm{d}y = \frac{1}{t}\int_0^{\infty} h\left(\frac{x-y}{t}\right) \,\mathrm{d}y$$

and so

$$(\rho_t)'(x) = \frac{1}{t^2}\int_0^{\infty} h'\left(\frac{x-y}{t}\right) \,\mathrm{d}y = -\frac{1}{t}h\left(\frac{x}{t}\right)$$

Based on this, we get an approximation $\nabla(j_t)_x$ of the gradient of the function j which is

$$\nabla(j_t)_x(x,\chi) = (r_1\chi_1, ..., r_n\chi_n)^T - d \cdot \left(-\frac{1}{t} \cdot h\left(\frac{g(x,\chi)-c}{t}\right) \cdot \chi \cdot (g(x,\chi)-c) + \mathbb{1}_{\mathbb{R}^+}(g(x,\chi)-c) \cdot \chi \right)$$

Various functions may be chosen for h. In (Andrieu et al., 2007) the authors study different such choices. For each one of them, they compute a reference value for the mean square error of the obtained approximated gradient. It turns out that, among the presented functions, $h := \frac{3}{4}(1-x^2)\mathbb{1}_1(x)$ is the best choice concerning this value (here $\mathbb{1}_1$ is the indicator function for the interval $]-1,1[$). This leads us to the following estimation of the gradient of j:

$$\nabla(j_t)_x(x,\chi) = (r_1\chi_1, ..., r_n\chi_n)^T + d \cdot \left(\frac{3}{4t}\left(1 - \left(\frac{g(x,\chi)-c}{t}\right)^2\right) \mathbb{1}_1\left(\frac{g(x,\chi)-c}{t}\right) \cdot \chi \cdot (g(x,\chi)-c) - \mathbb{1}_{\mathbb{R}^+}(g(x,\chi)-c) \cdot \chi \right)$$

3.1.2 The constrained knapsack problem

As presented in section 2, we consider two constrained knapsack problems, one with a chance and one an expectation constraint. As

$$\mathbb{P}\{g(x,\chi) \leq c\} = \mathbb{E}[\mathbb{1}_{\mathbb{R}^+}(c - g(x,\chi))]$$

these two considered variants of the stochastic knapsack problem are in fact equivalent. As in the case of $SRKP$, CKP has a deterministic equivalent formulation of the objective funtion:

$$\mathbb{E}[\sum_{i=1}^{n} r_i\chi_i x_i] = \sum_{i=1}^{n} r_i\mu_i x_i$$

The chance constrained knapsack problem

Generally, the chance constraint (3) does not define a convex set which makes the resolution even of continuous chance constrained problems difficult.

It has been shown by Prekopa (1995) that the set defined by constraint (3) is convex if χ has a log-concave density and g is quasi-convex. The first property can easily be proved for normal distributions and as our function g is linear, it is also (quasi-)convex. This means that the chance constraint (3) defines a convex set in the special case of a chance constrained knapsack problem with normally distributed weights.

We solve the continuous $CCKP$ by reformulating it as an equivalent, deterministic Second-order-cone-programming ($SOCP$) problem (Boyd et al., 1998). An $SOCP$ problem is an optimization problem of the following form:

$$\max_{x \in X_{ad}} \quad v^T x \tag{7}$$

$$\text{s.t. } \|Ax + b\| \leq c^T x + d \tag{8}$$

where $A \in \mathbb{R}^n \times \mathbb{R}^n$, $x, v, b, c \in \mathbb{R}^n$ and $d \in \mathbb{R}$. In the following, we call a constraint of the form (8) an *SOCP-constraint*.

Let Σ be the matrix of covariances of the probability vector χ. As we assume $p > 0.5$, we get the following equivalence (see e.g. (Boyd et al., 1998))

$$\mathbb{P}\{g(x,\chi) \leq c)\} \geq p \quad \Longleftrightarrow \quad \sum_i \chi_i x_i + F^{-1}(p)\|\Sigma^{1/2} x\| \leq c$$

Notice that Σ is a diagonal matrix as the weights are independently distributed. Therefore, its square root $\Sigma^{\frac{1}{2}}$ is also diagonal having the standard deviations of the random variables χ_i as nonzero diagonal components.

Based on this, the relaxed chance constrained knapsack problem becomes

$$\begin{array}{ll} & \max_{x \in [0,1]^n} \quad \mathbb{E}[\sum_i r_i \chi_i x_i] \\ \text{s.t.} & \sum_i \mu_i x_i + \delta \|\Sigma^{1/2} x\| \leq c \end{array} \quad \Leftrightarrow \quad \begin{array}{ll} & \max_{x \in [0,1]^n} \quad \mathbb{E}[\sum_i r_i \chi_i x_j] \\ \text{s.t.} & \|\Sigma^{1/2} x\| \leq -\frac{1}{\delta}\sum_i \mu_i x_i + \frac{c}{\delta} \end{array}$$

where $\delta := F^{-1}(p) > 0$.

The constraint $0 \leq x_i \leq 1$ $(i = 1, ..., n)$ of the corresponding relaxed problem can also be rewritten as an $SOCP$ constraint:

$$0 \leq x_i \leq 1 \Leftrightarrow \|A_i x\| \leq x_i \wedge \|A_i x\| \leq 1$$

where $A_i \in \mathbb{R}^{1 \times n}$, $A_i[1, k] = 0 \;\forall k \neq i$ and $A_i[1, i] = 1$.

Then, the $SOCP$ problem becomes:

$$\max_{x\in\mathbb{R}^n} \quad v^T x \tag{9}$$

$$\text{s.t.} \quad \|\Sigma^{1/2}x\| \le -\frac{1}{\delta}\cdot\mu\cdot x + \frac{c}{\delta} \tag{10}$$

$$\|A_i x\| \le \nu^i x \ , \tag{11}$$

$$\|A_i x\| \le 1 \ , \tag{12}$$

where $v := (r_1\mu_1, \ldots, r_n\mu_n)$ and $\nu^i \in \mathbb{R}^n$ such that ${\nu^i}_k = 1$ if $k = i$ and ${\nu^i}_k = 0$ otherwise.

This problem does not have any strictly feasible solution vector, as constraint (11) is always tight. This becomes problematic if we want to solve this problem using the *SOCP* program by Boyd, Lobo, Vandenberghe (Boyd et al. (1995)) as it applies an interior point method and can thus only solve strictly feasible problems. To get a strictly feasible solution, we perform a small perturbation on the right hand side of (11) by adding ε to $\nu^i x$ such that $0 < \varepsilon << 1$.

In order to solve problem (9)-(12) using the *SOCP* code by Boyd et al., we further determine its dual and check whether it is strictly feasible:

$$\begin{aligned}
&\min_{\substack{w^1\in\mathbb{R},\\ w^2,w^3,z^1,z^2,z^3\in\mathbb{R}^n}} && -\frac{c}{\delta}\cdot w^1 - \sum_{i=1}^n w_i^3 \\
&\text{s.t.} && \left(\Sigma^{1/2}\right)^T z^1 + \sum_{i=1}^n A_i^T(z_i^2 + z_i^3) - \frac{1}{\delta}\cdot\mu\cdot w^1 + \sum_{i=1}^n \nu^i w_i^2 = v \\
& && \|z^1\| \le w^1 \\
& && \|z_i^k\| \le w_i^k \ , \quad k = 2,3,\ i = 1,\ldots,n
\end{aligned}$$

where $z^1 \in \mathbb{R}^n, z_i^2, z_i^3 \in \mathbb{R}$ and $w^1, w_i^2, w_i^3 \in \mathbb{R}, i = 1,\ldots,n$, are the dual decision variables.

To find a strictly feasible solution vector for the dual problem, we reformulate it as follows:

$$\begin{aligned}
&\min_{\substack{w^1\in\mathbb{R},\\ w^2,w^3,z^1,z^2,z^3\in\mathbb{R}^n}} && -\frac{c}{\delta}\cdot w^1 - \sum_{i=1}^n w_i^3 \\
&\text{s.t.} && \sigma_i\cdot z_i^1 + z_i^2 + z_i^3 - \frac{\mu_i}{\delta}\cdot w^1 + w_i^2 = r_i\mu_i \ , \ i = 1,...,n \\
& && \sqrt{\sum_{i=1}^n (z_i^1)^2} \le w^1 \\
& && |z_i^k| \le w_i^k \ , \quad k = 2,3,\ i = 1,\ldots,n
\end{aligned}$$

At this point, we first use the fact that the random weights are independently distributed. If we choose $z^1 = z_i^k = 0$ $(k = 1,2\ ,\ i = 1,\ldots,n)$ and arbitrary $w_i^3 > 0$ $(i = 1,\ldots,n)$, it is easy to find strictly feasible w^1, w_i^2 $(i = 1,\ldots,n)$.

The expectation constrained knapsack problem

Generally, expectation constrained knapsack problems can be formulated as follows:

$$\max_{x\in\{0,1\}^n} \mathbb{E}[\sum_{i=1}^{n} r_i \chi_i x_i] \tag{13}$$

$$\text{s.t.} \quad \mathbb{E}[\Theta(x,\chi)] \leq \alpha \tag{14}$$

where $\alpha \in \mathbb{R}$ and $\Theta : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ is a function such that (14) represents the capacity constraint.

If constraint (14) is convex and Θ is differentiable, $ECKP$ can be solved by a stochastic Arrow-Hurwicz algorithm (see Algorithm 3.2). The stochastic Arrow Hurwicz algorithm is a stochastic gradient algorithm for solving constrained stochastic optimization problems by using Lagrangian multipliers.

Stochastic Arrow-Hurwicz Algorithm

1. Choose $x^0 \in X^{ad}$ and $\lambda^0 \in [0,\infty)$ as well as two α-sequences $(\epsilon^k)_{k\in\mathbb{N}}$ and $(\rho^k)_{k\in\mathbb{N}}$
2. Given x^k and λ^k, we draw χ_{k+1} following its normal distribution, we calculate $r^k = \nabla j(x^k, \chi^{k+1})$, $\theta^k = \nabla\Theta(x^k, \chi^{k+1})$ and we update x^{k+1} and λ^{k+1} as follows:

$$x^{k+1} = x^k + \epsilon^k(r^k + (\theta^k)^T \lambda^k)$$

$$\lambda^{k+1} = \lambda^k + \rho^k \Theta(x^{k+1}, \chi^{k+1})$$

3. For all $i = 1, ..., n$: If $x_i^{k+1} > 1$ set $x_i^{k+1} = 1$ and if $x_i^{k+1} < 0$ set $x_i^{k+1} = 0$
4. For all $i = 1, ..., n$: If $\lambda_i^{k+1} < 0$ set $\lambda_i^{k+1} = 0$

Algorithm 3.2:

As the set defined by the expectation constraint (5) is the same as the set defined by the chance constraint (3), it is also convex. With the approximation by convolution method showed in section 3.1.1, we can approximate the gradient of the constraint function $\mathbb{E}[\mathbb{1}_{\mathbb{R}^+}(c - \sum_i \chi_i x_i)]$. This allows to solve the $ECKP$ (4) using the stochastic Arrow-Hurwicz algorithm.

3.2 Calculating lower bounds

To calculate lower bounds on the solution value, we use a B&B algorithm based on an algorithm by Cohn and Barnhart (1998). In 3.2.1 we first explain and justify the ranking of the items using dominance relationships. Then, we present the B&B algorithm (see algorithm 3.3) and its variant for solving a CKP.

3.2.1 Ranking the items

In order to define the binary tree used in the B&B algorithm, we rank our items. We therefor introduce dominance relationships and the item are ranked according to the number of items they dominate and, in the case where several items dominate the same number of items, by their value of $\frac{r_i^2}{\sigma_i}$.

The dominance relationships are also used to prune subtrees during the algorithm in order to decrease the number of considered nodes and evaluated branches: whenever an item is rejected, we also reject all those items that are dominated by the rejected one.

$SRKP$**:** To introduce dominance relationships in the case of $SRKP$, we consider the variations of the (deterministic equivalent) objective function (6) J_{det}.

Clearly, the increase of one of the rewards per weight unit r_j increases the objective function if and only if $x_j > 0$.

To study the variations when changing the value of $\hat{\sigma}$, we calculate the derivative of J_{det} with respect to $\hat{\sigma}$:

$$\frac{\partial J_{det}}{\partial \hat{\sigma}}(x) = -d \cdot f\left(\frac{c - \hat{\mu}}{\hat{\sigma}}\right)$$

As f is strictly positive, this shows that whenever an item is replaced by another one having the same mean and reward per weight unit but smaller variance, the value of the objective function increases. Based on this study, Cohn and Barnhart (1998) introduced two types of dominance relationships: We say that item i dominates item k if one of the following holds:

1. $\mu_i = \mu_k$, $r_i \geq r_k$, $\sigma_i \leq \sigma_k$
2. $\mu_i \leq \mu_k$, $\sigma_i \leq \sigma_k$, $r_i \cdot \mu_i \geq r_k \cdot \mu_k$

CKP**:** In the case of $CCKP$ and $ECKP$, it is more complicated to introduce dominance relationships as in the case of $SRKP$. This is due to the fact that modifying $\hat{\sigma}$ cannot be interpreted as easily as in the former case. The only, very special case where one can say that item i dominates item k is the following:

1. $\mu_i = \mu_k$, $\sigma_i = \sigma_k$, $r_i >= r_k$

Most of the time, the items are thus simply ranked by their value of r_i. This ranking gave the best results in the numerical tests (compared e.g. with the ranking used for $SRKP$ or a random ranking) but can surely be improved.

3.2.2 The branch-and-bound algorithm

B&B algorithm 3.3 is based on the B&B algorithm by Cohn and Barnhart (1998). We just added step 4.

The algorithm has been constructed for $SRKP$. In order to use this algorithm to solve $CCKP$ or $ECKP$, we modify step 2 in order to respect the chance or the expectation

Branch-and-Bound Algorithm

1. Rank the items as described in section 3.2.1. This ranking defines the binary tree with the highest ranked item at the root.
2. Plunge the tree as follows: Beginning at the root of the tree, add the current item if and only if the objective function increases. Assign the maximum value of the objective function found to the variable INF. This variable stores the current lower bound of the objective function. Add the found branch to the list of branches. Set the associated upper bound SUP to infinity.
3. **If** there is no branch left on our list of branches, go to step 7.
   Else take the branch of our list of branches having the maximum objective function value. Go to step 4.
4. **If** the associated upper bound SUP is greater than the current lower bound INF, go to step 5.
   Else delete the branch from the list. Go to step 3.
5. **If** there is no accepted item left in the selected branch that does not already have a plunged or rejected subtree, delete the branch from the list. Go back to step 3.
   Else, following our ranking, choose the first accepted item that does not already have a plunged or rejected subtree. Calculate an upper bound SUP for the subtree defined by rejecting this item. Go to step 6.
6. **If** SUP $\leq$ INF, reject this subtree, go to step 5.
   Else plunge the subtree as described in 2 and add the found branch together with the value SUP to the list of branches. If the value of the objective function of this branch is greater than INF, update INF.
   Go to step 3.
7. The current value INF is the optimal solution of problem (1).

Algorithm 3.3:

constraint: instead of testing if the next item increases the objective function (which is the case for each item at every time), we check whether the chance or the expectation constraint is still satisfied when adding the next item. For example, in the case of $CCKP$, we calculate $\Phi(c)$ i.e. the cumulative distribution function of the probability variable $X = g(x, \chi)$. Then, depending on whether the obtained value is greater or equal than the prescribed probability, we accept or reject the item.

In step 5 the calculation of upper bounds for subtrees is realized by fixing the value of items that are higher in the tree at 1 or 0 and solving the continuous problem having the x_i of the remaining items as decision variables. In the case of $SRKP$ as well as $ECKP$ and the corresponding stochastic gradient algorithms this is easily done: at each iteration, we just leave out the recalculation of the fixed x_i. In the case of the $SOCP$ reformulation of $CCKP$, we solve the $SOCP$ subproblem with respect to the index set I of the items not already fixed (see subsection 3.1.2).

4 Numerical results

In this section we present our numerical results concerning the algorithms presented above. The first part contains the results of the algorithms for solving the continuous knapsack problems, namely the stochastic gradient method, the stochastic Arrow-Hurwicz algorithm as well as the algorithm by Boyd et al. to solve the $SOCP$ reformulation. The first two algorithms are implemented in C language. The third one is an open source interior point algorithm whose source code can be obtained as C- as well as $MATLAB$-code. We use the C-code. In the second part of the section, the results for the B&B algorithm are presented. It has also been implemented in C-language. All tests were carried out on an Intel PC with $1GB$ RAM.

Object	**reward per weight unit** r_i	**mean of the weight** μ_i	**variance** σ_i^2	$\frac{r_i^2}{\sigma_i}$
1	2	212	47	0.583
2	2	203	21	0.873
3	3	246	42	1.389
4	2	223	21	0.873
5	2	230	15	1.033
6	1	233	10	0.316
7	2	235	11	1.206
8	2	222	33	0.696
9	1	210	36	0.167
10	2	299	42	0.617
11	2	256	25	0.800
12	3	250	19	2.065
13	1	194	24	0.204
14	3	207	22	1.919
15	1	182	14	0.267

Table 1: Values of the Cohn-instance

We test our methods on the same dataset as in (Cohn and Barnhart, 1998) as well as a sample of randomly created instances for each of the chosen dimensions. The *Cohn-instance* is presented in Table 1. The last column states the value of the ratio r_i^2/σ_i used for the ranking of the items. The penalty factor used is 5. For the random datasets, the weight means are generated from a normal distribution with mean 225 and standard deviation 25, the variances from a uniform distribution on the interval $[5, 50]$ and the rewards per weight unit have equal probability to be 1, 2 or 3. In the case of $SRKP$, the penalty factor is always 5 and for $CCKP$ and $ECKP$ we choose a prescribed probability of $p = 0.6$. For each dimension we created 50 instances. Table 2, 3 and 4 show the average values over these 50 instances.

In the case of $SRKP$, our stochastic gradient algorithm and the corresponding B&B

algorithm are compared with the method of Cohn and Barnhart. In their paper, they propose three different upper bounds to use within their B&B algorithm. However, they do not give any details of which upper bound to use at which moment. In order to compare our approach with that of Cohn and Barnhart, we therefore compute their upper bounds one after another in step 5 of our B&B algorithm. As soon as an obtained bound is sufficiently tight to prune the currently evaluated subtree, we leave the computation of the remaining bounds out. This procedure assures that the number of considered nodes is at most as large as when using their exact policy.

4.1 The continuous stochastic knapsack problem

An example for the convergence of the stochastic gradient method involving approximation by convolution is shown in Figure 1. As shown in the figure and confirmed by numerical tests, the best result found does not change very much (less than 1%) after iteration 500. Based on this observation, we use in the following a stopping criterion for the stochastic gradient algorithm of 500 iterations.

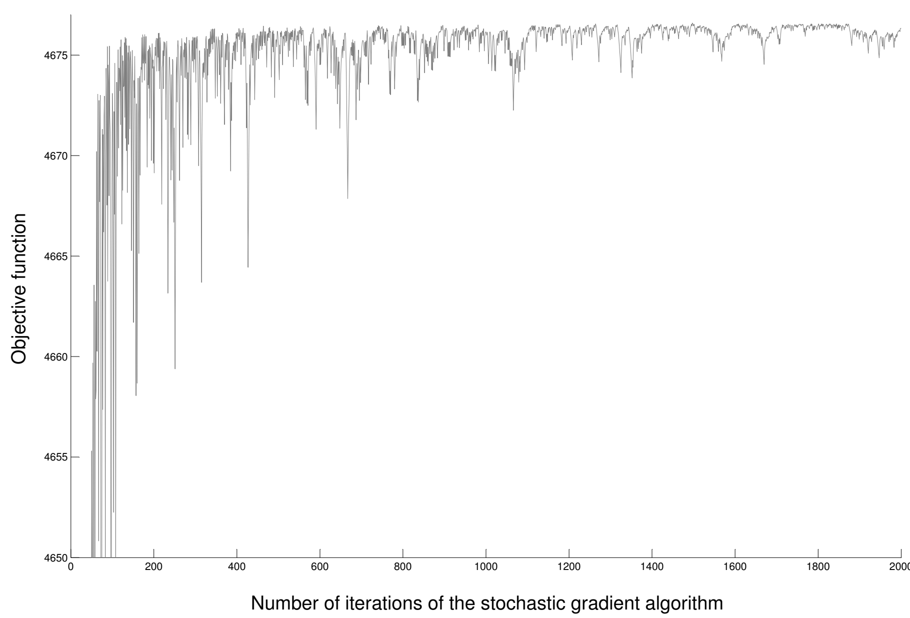

Figure 1: Results for the stochastic gradient algorithm solving the continuous $SRKP$

An example for the convergence of the stochastic Arrow-Hurwicz algorithm involving approximation by convolution is presented in Figure 2. The first graph shows the variations of the value of the objective function whilst the second figure presents the variations of the Lagrange multiplier λ. As in the case of the stochastic gradient algorithm, we fix a maximum number of 500 iterations for all further tests.

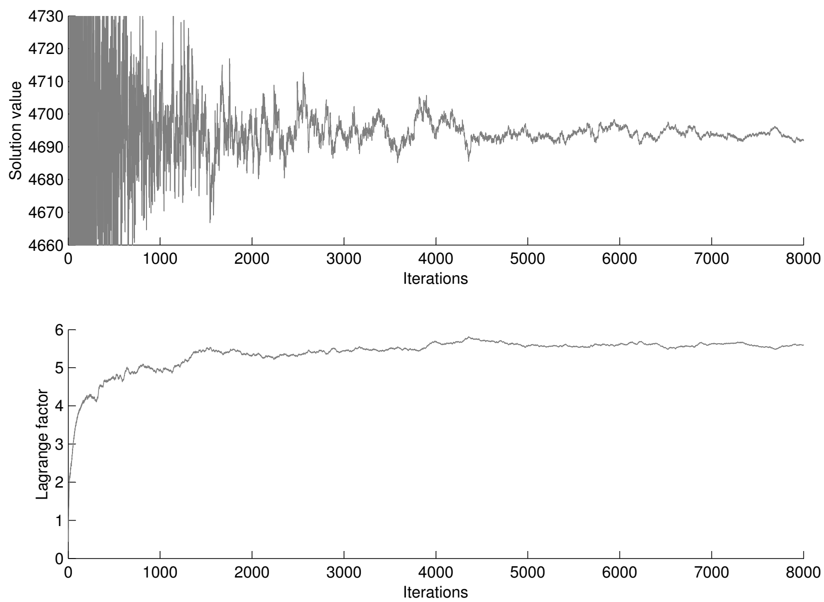

Figure 2: Results for the Arrow-Hurwicz algorithm solving the continuous $ECKP$

In Table 2 and Table 3 we compare for one thing the found optima of the continuous problems, or, more precisely, the calculated upper bounds for the combinatorial problem. For another, we compare the CPU time (in milliseconds) needed to compute them. C./B. stands for Cohn/Barnhart, i.e. for the (unique) Cohn-instance of dimension 15.

Table 2 gives the results for $SRKP$. We observe that especially for small dimensions it takes much less time to compute all three upper bounds proposed by Cohn and Barnhart than to solve the continuous relaxation by a stochastic gradient algorithm. But, while the CPU time of the stochastic gradient algorithm increases proportional to the dimension, this is not the case for the upper bounds proposed by Cohn and Barnhart.

Table 3 gives the results for CKP. As expected, the $SOCP$ algorithm solves the continuous CKP more accurate than the Arrow-Hurwicz algorithm, i.e. it finds a better solution value. Concerning the CPU time, the $SOCP$ algorithm needs as much time as the Arrow-Hurwicz algorithm to solve the continuous problems of very small dimension ($n = 15, 20$). However, for higher dimensional problems the Arrow-Hurwicz algorithm is much faster than the $SOCP$ method. The $SOCP$ algorithm is also very memory space consuming: for dimensions higher than $n = 180$ the memory space of the computer used is not sufficient to solve the continuous problem using the $SOCP$ program by Boyd et al..

	Stochastic gradient & Approx. by convolution		Cohn/Barnhart	
n	Optimum	CPU-time (msec)	Optimum	CPU-time (msec)
C./B.	4676.208	4	4759.000	< 1
15	4934.583	4	5146.927	< 1
20	6690.744	6	6936.017	< 1
30	10279.908	9	10529.541	< 1
50	16954.343	12	17224.803	< 1
75	25519.688	16	25811.775	< 1
100	33846.095	22	34131.754	< 1
150	50607.008	31	50932.104	< 1
250	85098.136	52	85459.649	1
500	170110.459	104	170503.708	3
1000	340922.966	240	340822.740	5
5000	1703811.095	1110	1704935.949	107
20000	6813327.586	4940	6815663.089	1759

Table 2: The numerical results for the continuous $SRKP$

4.2 The combinatorial stochastic knapsack problem

The numerical results for the combinatorial problem are shown in Table 4. Notice that the CPU time needed by the B&B algorithm (columns 6 and 11) is given in seconds. Columns 5 and 10 contain the number of considered nodes, i.e. the number of times an upper bound is calculated during the B&B algorithm.

The upper table of Table 4 contains the results for $SRKP$. We observe that when using the Cohn and Barnhart upper bounds during the B&B algorithm much more nodes have to be considered. This can be explained by the less tighter upper bounds and, consequently, a smaller number of rejected subtrees. For small dimensions ($n = 15, 20, 30$) this is counterbalanced by the small CPU times needed to calculate one upper bound. In the case of higher dimensional problems, the B&B algorithm involving a stochastic gradient algorithm becomes more competitive due to the tighter upper bounds and the resulting smaller number of considered nodes.

Studying the lower table in Table 4, we observe that when using the Arrow-Hurwicz algorithm a smaller number of nodes has to be considered to solve CKP than with the $SOCP$ program. This is not, as in the case of $SRKP$, due to a better choice of the upper bounds as in both algorithms the upper bounds are supposed to be the solution of the relaxed problem. Nevertheless, we get smaller values when calculating them using the Arrow-Hurwicz algorithm. This is based on the fact that the Arrow-Hurwicz algorithm involving approximation by convolution only computes approximate solutions of the relaxed problems. These non-optimal solutions have, of course, a smaller value than the optimum and the chosen "upper bounds" seem to be tighter. As the duality gaps of the chosen instances are very small, these smaller "upper bounds" have a great impact, i.e. a lot

	Arrow-Hurwicz & Approx. by convolution		SOCP	
n	**Optimum**	**CPU-time (msec)**	**Optimum**	**CPU-time (msec)**
C./B.	4696.097	3	4696.413	4
15	4954.546	4	4954.704	4
20	6713.081	5	6713.987	6
30	10308.640	7	10310.45	18
50	16992.450	11	16993.514	65
75	25568.059	17	25569.379	213
100	33902.283	22	33903.672	503
150	50676.686	32	50678.312	1802
250	85187.249	52	**	**
500	170239.531	107	**	**
1000	340529.019	216	**	**
5000	1704560.250	1100	**	**
20000	6814158.873	4317	**	**

** exceeding of the available memory space

Table 3: The numerical results for the continuous $CCKP/ECKP$

more subtrees are rejected. This can theoretically also cause the exclusion of a subtree that contains the optimal solution. Anyway, in the case of our instances, the found optima are in both cases nearly the same.

As mentioned, Table 4 only shows the results for the combinatorial problem in the case where the average needed time over all 50 instances is at most 1h. In case of the stochastic gradient algorithm involving approximation by convolution, this limit is respected when $n = 75$ but exceeded when $n = 100$. For $n = 100$, the the CPU-time is smaller or equal than $2h$ in about 78% of the cases and only 6% of the instances needed more than $24h$ to terminate. For $n = 150$, 44% of the tests finished in at most $2h$ and 56% of the instances needed not more than $24h$.

5 Conclusion

In this paper we study, solve and compare two different variants of a stochastic knapsack problem with random weights. We apply a B&B algorithm and solve continuous subproblems in order to provide upper bounds. We use a stochastic gradient method for solving the continuous stochastic knapsack problem with simple recourse ($SRKP$) and an $SOCP$ algorithm as well as a stochastic Arrow-Hurwicz algorithm for solving the constrained version of the continuous stochastic knapsack problem (CKP). In the cases of the stochastic gradient and the Arrow-Hurwicz algorithms, approximated gradients are computed using

	Stochastic gradient & Approximation by convolution					Cohn/Barnhart				
n	**Upper Bound**	**CPU-time (msec) continuous**	**Optimum**	**considered nodes**	**CPU-time (sec) B-and-B**	**Upper Bound**	**CPU-time (msec) continuous**	**Optimum**	**considered nodes**	**CPU-time (sec) B-and-B**
C./B.	4676.208	4	4618	100	0.342	4759.000	< 1	4618	144	0.000
15	4934.583	4	4890	41	0.139	5146.927	< 1	4890	65	0.002
20	6690.744	6	6651	80	0.348	6936.017	< 1	6651	280	0.003
30	10279.908	9	10265	455	2.808	10529.541	< 1	10265	2525	0.037
50	16954.343	12	16951	13173	131.171	17224.803	< 1	16951	364960	779.325
75	25519.688	16	25514	63972	934.550	25811.775	< 1	*	*	*
100	33846.095	22	*	*	*	34131.754	< 1	*	*	*

* CPU-time exceeds 1h

	Arrow-Hurwicz & Approximation by convolution					SOCP				
n	**Upper Bound**	**CPU-time (msec) continuous**	**Optimum**	**considered nodes**	**CPU-time (sec) B-and-B**	**Upper Bound**	**CPU-time (msec) continuous**	**Optimum**	**considered nodes**	**CPU-time (sec) B-and-B**
C./B.	4696.097	3	4595	122	0.469	4696.413	4	4595	122	0.406
15	4954.546	4	4840	34	0.116	4954.704	4	4840	34	0.082
20	6713.081	5	6634	71	0.305	6713.987	6	6634	66	0.236
30	10308.640	7	10272	345	2.314	10310.45	18	10272	350	1.801
50	16992.450	11	16974	1880	19.473	16993.514	65	16975	7406	70.914
75	25568.059	17	25547	3743	57.397	25569.379	213	25548	62175	1535.520
100	33902.283	22	33893	94984	1932.097	33903.672	503	*	*	*
150	50676.686	32	*	*	*	50678.312	1802	*	*	*

* CPU-time exceeds 1h

Table 4: Numerical results for the combinatorial $SRKP$ (upper table) and $CCKP/ECKP$ (lower table)

approximation by convolution.

Concerning $SRKP$, we compare the B&B algorithm involving the stochastic gradient method with a method from literature (Cohn and Barnhart (1998)). Our numerical tests show, that our upper bounds are tighter, i.e. less nodes have to be considered. This results for higher dimensional problems in smaller CPU times. In the case of CKP, the Arrow-Hurwicz algorithm shows a better performance for large size instances as the time to compute one upper bound is smaller. In addition, the $SOCP$ algorithm can not solve large instances due to its higher memory requirements which results in an exceeding of the available memory space for high dimensional problems.

References

Andrieu, L., Cohen, G., and Vzquez-Abad, F. (2007). Stochastic programming with probability constraints. http://fr.arxiv.org/abs/0708.0281 (Accessed 24 October 2008).

Ağralı, S. and Geunes, J. (2008). A class of stochastic knapsack problems with poisson resource requirements. http://plaza.ufl.edu/sagrali/research_files/Poisson_KP_ORL_Submission.pdf (Accessed 24 October 2008).

Babaioff, M., Immorlica, N., Kempe, D., and Kleinberg, R. (2007). A knapsack secretary problem with applications. In *APPROX-RANDOM*, pages 16–28.

Boyd, S., Lebret, H., Lobo, M. S., and Vandenberghe, L. (1998). Applications of second-order cone programming. *Linear Algebra and its Applications*, 284:193–228.

Boyd, S., Lobo, M. S., and Vandenberghe, L. (1995). Software for second-order cone programming. http://www.stanford.edu/~boyd/old_software/socp/doc.pdf (Accessed 24 October 2008).

Boyd, S. and Vandenberghe, L. (2004). *Convex Optimization*. Cambridge University Press.

Carraway, R. L., Schmidt, R. L., and Weatherford, L. R. (1993). An algorithm for maximizing target achievement in the stochastic knapsack problem with normal returns. *Naval research logistics*, 40(2):161–173.

Claro, J. and de Sousa, J. P. (2008). A multiobjective metaheuristic for a mean-risk static stochastic knapsack problem. www.springerlink.com/index/b668736740218j72.pdf (Accessed 24 October 2008).

Cohn, A. and Barnhart, C. (1998). The stochastic knapsack problem with random weights: A heuristic approach to robust transportation planning. In *Proceedings of the Triennial Symposium on Transportation Analysis (TRISTAN III)*.

Dean, B. C., Goemans, M. X., and Vondrák, J. (2004). Approximating the stochastic knapsack problem: The benefit of adaptivity. In *Proceedings 45th Annual IEEE 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 208–217.

Ermoliev, Y. M., Norkin, V. I., and Wets, R. J.-B. (1995). The minimization of semicontinuous functions: Mollifier subgradients. *SIAM Journal on Control and Optimization*, 33(1):149–167.

Goel, A. and Indyk, P. (1999). Stochastic load balancing and related problems. In *40th Annual Symposium on Foundations of Computer Science*, pages 579 – 586.

Harvey M. Salkin, C. A. D. K. (2006). The knapsack problem: A survey. *Naval Research Logistics*, 22(1):127–144.

Henig, M. I. (1990). Risk criteria in a stochastic knapsack problem. *Operations Research*, 38(5):820–825.

Kellerer, H., Pferschy, U., and Pisinger, D. (2004). *Knapsack Problems*. Springer-Verlag (Berlin, Heidelberg).

Kieper, J. and Wolfowitz, J. (1952). Stochastic estimation of the maximum of a regression function. *Annals of Mathematical Statistics*, 23:462–466.

Kleinberg, J., Rabani, Y., and Tardos, E. (1997). Allocating bandwidth for bursty connections. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 664 – 673.

Kleywegt, A. J. and Papastavrou, J. D. (2001). The dynamic and stochastic knapsack problem with random sized weights. *Operations Research*, 49:26–41.

Kleywegt, A. J., Shapiro, A., and de mello, T. H. (2001). The sample average approximation method for stochastic discrete optimization. *SIAM Journal on Optimization*, 12:479–502.

Klopfenstein, O. and Nace, D. (2006). A robust approach to the chance-constrained knapsack problem. http://www.optimization-online.org/DB_HTML/2006/03/1341.html (Accessed 24 October 2008).

Kolesar, P. J. (1967). A branch and bound algorithm for the knapsack problem. *Management Science*, 13(9):723–735.

Kushner, H. J. and Yin, G. G. (2003). *Stochastic Approximation and Recursive Algorithms and Applications*. Springer Verlag.

L'Écuyer, P. and Yin, G. (1998). Budget dependent convergence rate of stochastic approximation. *SIAM Journal Optimization*, 8:217–247.

Lin, G., Lu, Y., and Yao, D. (2008). The stochastic knapsack revisited: Switch-over policies and dynamic pricing. *Operations Research*, 56:945–957.

Lu, Y. (2008). Approximating the value functions of stochastic knapsack problems: a homogeneous monge-ampère equation and its stochastic counterparts. http://arxiv.org/abs/0805.1710 (Accessed 24 October 2008) (to be published in International Journal of Mathematics and Statistics).

Marchetti-Spaccamela, A. and Vercellis, C. (1995). Stochastic on-line knapsack problems. *Mathematical Programming*, 68:73–104.

Martello, S. and Toth, P. (1977). An upper bound for the zero-one knapsack problem and a branch and bound algorithm. *European Journal of Operational Research*, 1(3):169–175.

Morton, D. P. and Wood, R. K. (1997). *Advances in Computational and Stochastic Optimization, Logic Programming and Heuristic Search*, chapter On a stochastic knapsack problem and generalizations, pages 149–168. Kluwer Academic Publishers (Norwell, MA, USA).

Nevel'son, M. B. and Has'minskii, R. Z. (1976). *Stochastic Approximation and Recursive Estimation*. American Mathematical Society.

Polyak, B. T. (1990). New method of stochastic approximation type. *Automation and Remote Control*, 51:937–946.

Prekopa, A. (1995). *Stochastic Programming*. Kluwer Academic Publishers (Dordrecht, Boston).

Robbins, H. and Monro, S. (1951). A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407.

Ross, K. W. and Tsang, D. H. K. (1989). The stochastic knapsack problem. *IEEE Transactions on Communications*, 37(7):740–747.

Sahinidis, N. V. (2004). Optimization under uncertainty: state-of-the-art and opportunities. *Computers and Chemical Engineering*, 28:971983.

Sun, X., Sheng, H., and Li, D. (2007). An exact algorithm for 0-1 polynomial knapsack problems. *Journal of industrial and management optimization*, 3(2).