Pesquisa Operacional (2022) 42: e257386 p.1-33
doi: 10.1590/0101-7438.2022.042.00257386

Printed version ISSN 0101-7438 / Online version ISSN 1678-5142
www.scielo.br/pope
ARTICLES

A HYBRIDIZED MULTI-OBJECTIVE MEMETIC ALGORITHM FOR THE MULTI-OBJECTIVE STOCHASTIC QUADRATIC KNAPSACK PROBLEM

Amina Guerrouma[1] and Méziane Aïder[2*]

Received October 16, 2021 / Accepted April 19, 2022

ABSTRACT. The knapsack problem is basic in combinatorial optimization and possesses several variants and expansions. In this paper, we focus on the multi-objective stochastic quadratic knapsack problem with random weights. We propose a Multi-Objective Memetic Algorithm With Selection Neighborhood Pareto Local Search (MASNPL). At each iteration of this algorithm, crossover, mutation, and local search are applied to a population of solutions to generate new solutions that would constitute an offspring population. Then, we use a selection operator for the best solutions to the combined parent and offspring populations. The principle of the selection operation relies on the termination of the non-domination rank and the crowding distance obtained respectively by the Non-dominated Sort Algorithm and the Crowding-Distance Computation Algorithm. To evaluate the performance of our algorithm, we compare it with both an exact algorithm and the NSGA-II algorithm. Our experimental results show that the MASNPL algorithm leads to significant efficiency.

Keywords: non-dominated sort algorithm, crowding-distance, gradient algorithm, memetic algorithm with selection neighborhood pareto local search.

1 INTRODUCTION

Knapsack problems are widely studied general combinatorial optimization problems that are useful models for modeling many real-world problems in a variety of fields. The canonical version (KP) of this class of problems is NP-hard and, given a set of items, each with positive weight and profit, and a knapsack of fixed positive capacity, it consists of choosing a subset of items that fit in the knapsack and of maximum total profit. For instance, suppose a firm has a fixed budget for investment in several projects, each of which has a budget, and its implementation involves a payoff. The problem for decision-makers is to select a subset of projects to invest in so that

*Corresponding author
[1]LaROMaD, Fac. Maths, USTHB, PB 32 Bab Ezzouar, 16111 Algiers, Algeria – E-mail: amina.g2008@hotmail.com – http://orcid.org/0000-0003-2625-3973
[2]LaROMaD, Fac. Maths, USTHB, PB 32 Bab Ezzouar, 16111 Algiers, Algeria – E-mail: meziane.aider@usthb.edu.dz – http://orcid.org/0000-0001-7195-810X

the total profit of all chosen projects is maximum and the total budget invested in them does not exceed the available budget.

Two fine monographs are entirely devoted to knapsack problems in their different versions and variants. The first one, by Martello & Toth (1990), explicitly describes and comments in great detail on the algorithms and computer implementations of knapsack problems and contains, on 250 pages with 200 references, the main results in this field published during the three previous decades. The second monograph, by Kellerer et al. (2004), contains more than 500 pages, with about 500 references, two-thirds of which were published after the first monograph, which shows the interest of researchers in this field. In addition to these fine works, surveys are regularly published to update knowledge in the field. Cacchiani et al. (2022a,b), have recently proposed a survey on knapsack problems and structured it into two parts. Part I is devoted to problems whose goal is to optimally assign items to a single knapsack and review problems with special constraints and relatively recent fields of investigation, like robust and bilevel problems. Part II covers multiple, multidimensional, and quadratic knapsack problems, and includes a succinct treatment of online and multiobjective knapsack problems.

Here we focus on the quadratic form of KP, known as the Quadratic Knapsack Problem (QKP). Given a knapsack with a fixed capacity and a set of items, each item associated with a positive integer weight, an individual profit, and a paired profit achieved if both items are selected, we aim to choose a subset of items whose overall weight does not exceed the given knapsack capacity that maximizes the overall profit. A well-known survey by Pisinger (2007) gave numerous details about the problem and the resolution aspects. Gallo et al. (1980) introduced the quadratic knapsack problem and derived the first bounds by using the concept of "upper plane". The authors presented and discussed different uses of this concept in a branch-and-bound scheme for solving such a problem. Aïder et al. (2022a) considered a branch and solve strategies-based algorithm for solving large-scale quadratic multiple knapsack problems. They developed an enhanced fix and solve solution procedure and embedded it in the local branching-based method, where the branches reflect intensification and diversification search around a solution. The same authors Aïder et al. (2022b) used the hybrid population-based algorithm (namely HBPA) to efficiently solve the bi-objective quadratic multiple knapsack problem. Létocart et al. (2014) proposed, to solve the 0-1 exact k-item quadratic knapsack problem, a fast and efficient heuristic method that produces both good lower and upper bounds on the value of the problem in a reasonable time. Specifically, it integrates a primal heuristic and a semidefinite programming reduction phase within a surrogate dual heuristic. To solve the generalized quadratic multiple knapsack problem (GQMKP), Zhou et al. (2022) have proposed an efficient hybrid evolutionary search algorithm (HESA) that relies on a knapsack-based crossover operator to generate new offspring solutions, as well as a feasible and infeasible adaptive tabu search to improve the new offspring solutions. Other new features of HESA include a dedicated strategy to ensure a diversified and a high-quality initial population and a streamlining technique to speed up the evaluations of candidate solutions

However, in real life, problems are non-deterministic, that is, some parameters are known without precision at the time a decision must be made. These parameters can be modeled by continuously or discretely distributed random variables, which turn the underlying problem into a stochastic optimization problem. In this case, we refer to the work of Kosuch & Lisser (2011), who considered two models of stochastic knapsack problems with random weights. The first model is an unconstrained problem, namely the Stochastic Knapsack Problem With Simple Recourse. The authors proposed different solving methods for the corresponding relaxed problem as the branch-and-bound algorithm, while the second, as a two or multi-stage problem that allows later corrections of the decision made in advance. The authors proposed a method to calculate the upper and lower bounds. These bounds are used in the branch-and-bound framework. Kosuch et al. (2017) studied the stochastic knapsack problem with expectation constraints and proposed to solve the problem with the relaxed version of this problem using a stochastic gradient algorithm to provide upper bounds for a branch-and-bound framework. Blado & Toriello (2021) considered a two-stage stochastic multiple knapsack problem together with a set of possible disturbances with known probability of occurrence and proposed two branch-and-price approaches to solve it. For the same problem, Tönissen et al. (2021) used branch-and-price scheme and compared two different decomposition approaches. Range et al. (2018) considered the stochastic knapsack problem and combined the chance-constrained knapsack problem and the stochastic knapsack problem with simple recourse. They formulated the resulting model as a network problem and showed that it could be solved by a dynamic labeling programming approach for the shortest path problem with resource constraints. Tönissen & Schlicher (2021) introduced the two-stage stochastic 3D printing knapsack problem for which they provided a two-stage stochastic programming formulation which they later reformulated into an equivalent integer linear program. Song et al. (2018) studied the stochastic quadratic multiple knapsack problem with the objective to find a solution with the best expected quality under all possible cases. The authors used the recoverable robustness technique which consists of first finding a solution that is feasible for the static associated problem, and then adjusting the solution according to the emerging stochastic scenarios.

In this paper, we study the case where the item weights are random and are supposed to be independent, normally distributed, with known mean and variance while the capacity and benefits remain deterministic. This case entails the following problem: we cannot be sure that the chosen items in advance (i.e., before the revelation of the actual weights) meet the knapsack capacity constraint. We used a stochastic gradient algorithm via a convolution method, and we refer to Andrieu et al. (2007) and Kosuch & Lisser (2009).

The Stochastic Quadratic Knapsack Problem (SQKP) is NP-hard. For solving such a problem, exact and heuristic algorithms constitute two main and complementary solution methods in the literature. On the one hand, the computing time needed to find an optimal solution by an exact algorithm may become prohibitive for large instances. On the other hand, heuristic algorithms aim to find satisfactory sub-optimal solutions (for too large problem instances) in an acceptable computing time. For solving the Stochastic Quadratic Knapsack, Lisser & Lopez (2010) used the

solution techniques based on semi-definite relaxations. Song et al. (2018) studied the stochastic quadratic multiple knapsack problem with the objective to find a solution with the best expected quality under all possible cases. The authors used the recoverable robustness technique which consists to first find a solution that is feasible for the static associated problem, and then adjust the solution according to the emerging stochastic scenarios.

Most real-world problems like scheduling, resource allocation,... have to simultaneously optimize several criteria. These are called multi-objective optimization problems (MOOPs) and can, in mathematical terms, be formulated as:

$$\begin{aligned} \max F(x) &= (f_1(x), \ldots, f_m(x))^T \\ \text{subject to} \quad & x \in \Omega, \end{aligned} \tag{1}$$

where the integer $m \geq 2$ is the number of objectives and the set Ω is the feasible set of decision vectors, which is typically $\Omega \subseteq \mathbb{R}^n$.

Our work is mainly inspired by the work of Chen & Hao (2016) who extended the single objective quadratic multiple knapsack problem to a bi-objective quadratic multiple knapsack problem BO-QMKP and proposed a hybrid two-stage HTS algorithm to approximate the Pareto front.

We consider a scenario where given the solutions in some space (of possible solutions), the so-called decision space, which can be evaluated using the so-called objective functions, the goal is to find a solution which the decision-maker can agree, and that is optimal in some sense. Let $u, v \in \mathbb{R}^m$, u dominates v if $f_j(u) \geq f_j(v)$, for all objective j and $f_j(u) > f_j(v)$ for at least one objective $j \in \{1, \ldots, m\}$. A solution $x^* \in \Omega$ is Pareto Optimal (or efficient) to (1) if there is no point $x \in \Omega$ such that x dominates x^*. $F(x^*)$ is then called a Pareto Optimal (objective) vector. In other words, any improvement in a Pareto optimal point in one objective must lead to a deterioration in at least one other objective. The set of all the Pareto optimal points is called the Pareto set *PS*, and the set of all the Pareto optimal objective vectors is the Pareto front PF. It is well-known that a Pareto optimal solution to a MOP, under conditions, could be an optimal solution for a scalar optimization problem in which the objective is an aggregation of all the objective functions. Therefore, an approximation of PF can be decomposed into several scalar objective optimization subproblems. This is a basic idea behind many traditional mathematical programming methods for approximating PF.

Classical optimization methods (including multi-criteria decision-making methods) suggest converting a multi-objective optimization problem into a single-objective optimization problem to solve it. We were inspired by the methods used to solve the Multi-objective Capacitated Arc Routing Problem (MO-CARP), for which Tang et al. (2009) proposed a memetic algorithm with extended neighborhood search (MAENS), and Mei et al. (2011) developed a new Memetic Algorithm (MA) called Decomposition-based Memetic Algorithm with Extended Neighborhood Search D-MAENS. Shang et al. (2014) proposed another version of D-MAENS with two improvements consisting on the one hand in accelerating the convergence speed by the replacement of the solutions immediately once an offspring is generated,referring to the steady-state evo-

lutionary algorithm, and on the second hand in implementing elitism by using an archive to maintain the current best solution in its decomposition direction during the search.

Zhang & Li (2007) suggested a multi-objective evolutionary algorithm based on decomposition MOEA-D, that decomposes a multiobjective optimization problem into many scalar optimization subproblems and optimizes them simultaneously, each subproblem being optimized by only using information from its several neighboring subproblems. Bhuvana & Aravindan (2016) introduced a Preferential Local Search mechanism to fine-tune the global optimal solutions further, and an adaptive weight mechanism for combining multiple objectives. Kim & Liou (2012) proposed a novel fitness sharing method for Multi-Objective Genetic Algorithm by combining a new sharing function and sided degradations in the sharing process, with preference to either of two close solutions. Arshad et al. (2009) presented a sequence based genetic algorithm, for the symmetric traveling salesman problem, where a set of sequences are extracted from the best individuals, which are used to guide the search and some procedures are applied to maintain the diversity by breaking the selected sequences into sub tours if the best individual of the population does not improve.

Some of these ideas have been integrated into NSGA-II to get at a new memetic algorithm for solving multi-objective optimization problems. Deb et al. (2002) suggested a non-dominated sorting-based multi-objective evolutionary algorithm MOEA, called non-dominated sorting genetic algorithm NSGA-II, and Chu & Yu (2018) proposed the crowding distance in the standard NSGA-II.

The remainder of the paper is organized as follows. Section 2 presents a mathematical formulation of the quadratic multi-objective stochastic knapsack problem with simple recourse. Section 3 then describes, in detail, our solution methods that aim to provide a high-quality approximation of the Pareto front. We first introduce the NSGA-II algorithm, then sketch the greedy algorithm and the principle of subpopulation construction before describing the steps of the non-dominated sorting algorithm and detailing the crowding-distance algorithm. In the next subsection, we describe the memetic algorithm with local Pareto search by selection neighborhood. The numerical performance of our proposed method is analyzed and evaluated in Section 4, before concluding in Section 5.

2 MATHEMATICAL FORMULATION

We consider a stochastic quadratic multi-objective knapsack problem of the following form: given a knapsack with a fixed weight capacity $c > 0$ as well as a set of n items, $i = 1, \ldots, n$, each item has a weight that is not known in advance, i.e. the decision of which items to choose must be made without the exact knowledge of their weights. Therefore, we treat the weights as random variables and assume that the weights χ_i, $i = 1, \ldots, n$, are independent and normally distributed with means $\mu_i > 0$, and standard deviations $\sigma_i, i = 1, \ldots, n$. Moreover, each item $i = 1, \ldots, n$ has a fixed m-vector reward per weight unit $r_i = (r_i^1, \ldots, r_i^m)^T$, $r_i^k \in \mathbb{Z}^+, k = 1, \ldots, m$, and to each pair of items i and j, $1 \leq i \neq j \leq n$, is associated a m-vector of joint rewards per unit of weight

$r_{ij} = (r_{ij}^1, \ldots, r_{ij}^m)^T$, $r_{ij}^k \in \mathbb{Z}^+, k = 1, \ldots, m$. The choice of a reward per unit weight can be justified by the fact that the value of an item often depends on its weight, which is not known in advance.

In case of overweight, items must be removed, and a penalty d must be paid for each unit of weight unwrapped. Our goal is, therefore, to minimize the total penalty.

The selection of the item i is indicated by a binary decision variable x_i which takes the value 1 if item i is included in the selection and 0 otherwise.

The Multi-objective Stochastic Quadratic Knapsack Problem can be mathematically formulated as follows:

2.1 Stochastic Quadratic Knapsack Problem with simple recourse

$$\max_{x\in\{0,1\}^n} \mathbf{E}\Big[\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} r_{ij}^{(k)} x_i x_j(\chi_i+\chi_j)\Big] + \mathbf{E}\Big[\sum_{i=1}^{n} r_i^{(k)} x_i\chi_i\Big] - d\mathbf{E}\Big[\sum_{i=1}^{n} x_i\chi_i - c\Big]^+, k = 1, \ldots, m. \tag{2}$$

Equation (2) aims to maximize the total profit of all assigned objects.

The special case when $k = 2$ is called bi-objective binary knapsack problem and is denoted by 0-1 BOKP.

2.2 Constrained Knapsack Problem

A. Expectation Constrained Knapsack Problem

$$\begin{cases} \max\limits_{x\in\{0,1\}^n} \mathbf{E}\Big[\sum\limits_{i=1}^{n-1}\sum\limits_{j=i+1}^{n} r_{ij}^k x_i x_j(\chi_i+\chi_j)\Big],\ k = 1, \ldots, m & (3.1) \\ \text{s.t.} \quad \mathbf{E}\big[\mathbf{1}_{\mathbb{R}^+}\big(c - g(x,\chi)\big)\big] \geq p. & (3.2) \end{cases} \tag{3}$$

B. Chance Constrained Knapsack Problem

$$\begin{cases} \max\limits_{x\in\{0,1\}^n} \mathbf{E}\Big[\sum\limits_{i=1}^{n-1}\sum\limits_{j=i+1}^{n} r_{ij}^k x_i x_j(\chi_i+\chi_j)\Big],\ k = 1, \ldots, m & (4.1) \\ \text{s.t.} \quad \mathbf{P}\big[g(x,\chi) \leq c\big] \geq p, & (4.2) \end{cases} \tag{4}$$

where:

- $\mathbf{P}[A]$ denotes the probability of an event A,
- $\mathbf{E}[.]$ denotes the expectation,
- $\mathbf{1}_{\mathbb{R}^+}$ denotes the indicator function of the positive real interval,
- $g(x,\chi) = \sum\limits_{i=1}^{n} x_i\chi_i$,

- $d \in \mathbb{R}^+$,
- $p \in [0.5, 1]$ is the prescribed probability.

In the formulation of the multi-objective stochastic quadratic knapsack problem with simple recourse, the capacity constraint has been included in the objective function by using the penalty function $[.]^+$ and a penalty factor $d > 0$. In the case of an overload, items have to be removed and a penalty d has to be paid for each unit of weight that is unpacked.

We write the objective function of the Stochastic Quadratic Knapsack Problem with simple recourse as follows:

$$\mathbf{J}^{(k)}(x,\chi) = \mathbf{E}\Big[\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} r_{ij}^{(k)} x_i x_j(\chi_i+\chi_j)\Big] + \mathbf{E}\Big[\sum_{i=1}^{n} r_i^{(k)} x_i \chi_i\Big] - d\mathbf{E}\Big[\sum_{i=1}^{n} x_i\chi_i - c\Big]^+,\ k = 1,\ldots,m. \tag{5}$$

Since the function $\mathbf{J}$ is not differentiable, we present an approximation to its gradient, named approximation by convolution. This is one of the two methods presented by Andrieu et al. (2007).

The basic idea of this method called,"Approximation By Convolution Method" is to approximate the indicator function $(\mathbf{1}_{\mathbb{R}^+})$ by its convolution with a function :

$$h_t(x) := \frac{1}{t} h\Big(\frac{x}{t}\Big) \tag{6}$$

that approximates the *Dirac Function* when the parameter t goes to 0.

Let us consider a function $h : \mathbb{R} \longrightarrow \mathbb{R}$ with the following properties:

$$\begin{aligned} &\text{a) } h \text{ has a unique maximum at } x = 0,\\ &\text{b) } \forall x \in \mathbb{R},\ \ h(x) \geq 0,\\ &\text{c) } \forall x \in \mathbb{R},\ \ h(x) = h(-x),\\ &\text{d) } \int_{-\infty}^{+\infty} h(x)dx = 1. \end{aligned} \tag{7}$$

With any other function $\rho : \mathbb{R} \longrightarrow \mathbb{R}$ and a small positive number t, the convolution of the two functions ρ and h is defined as follows:

$$(\rho.h_t)(x) = \frac{1}{t}\int_{-\infty}^{+\infty} \rho(y) h\Big(\frac{y-x}{t}\Big)dy. \tag{8}$$

The function $(\rho.h_t)$ is differentiable with:

$$(\rho.h_t)'(x) = \frac{1}{t^2}\int_{0}^{+\infty} \rho(y) h'\Big(\frac{x-y}{t}\Big)dy = \frac{1}{t} h\Big(\frac{x}{t}\Big). \tag{9}$$

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

$$(\rho.h_t)(x) = \frac{1}{t}\mathbf{E}\Big[\int_{-\infty}^{+\infty} \mathbf{1}_{\mathbb{R}^+}(y)h\Big(\frac{x-y}{t}\Big)dy\Big] = \frac{1}{t}\mathbf{E}\Big[\int_{0}^{+\infty} h\Big(\frac{y-x}{t}\Big)dy\Big] \tag{10}$$

$$(\rho.h_t)(x) = \mathbf{E}\big[\rho_t(x)\big] \tag{11}$$

$$\rho_t(x) = \frac{1}{t}\mathbf{E}\Big[\int_{0}^{+\infty} h\Big(\frac{y-x}{t}\Big)dy\Big] \tag{12}$$

Then,

$$(\rho_t(x))'(x) = \frac{1}{t^2}\int_{0}^{+\infty} h'\Big(\frac{x-y}{t}\Big)dy = -\frac{1}{t}h\Big(\frac{x}{t}\Big). \tag{13}$$

Based on the method explained above, we get the following approximation $\nabla(\mathbf{J}_t)$ of the gradient of the function $\mathbf{J}$:

$$\begin{aligned}\nabla(\mathbf{J}_t^{(k)})(x,\chi) = \Big[(r_1^{(k)}\chi_1,\ldots,r_n^{(k)}\chi_n)^T + \Big(\sum_{\substack{j=1\\ j\neq 1}}^{n} r_{1j}^{(k)}(\chi_1+\chi_j)x_j,\ldots,\sum_{\substack{j=1\\ j\neq n}}^{n} r_{nj}^{(k)}(\chi_n+\chi_j)x_j\Big)^T\Big]\\ -d\Big(-\frac{1}{t}h\Big(\frac{g(x,\chi)-c}{t}\Big).\chi.\big(g(x,\chi)-c\big) + \mathbf{1}_{\mathbb{R}^+}\big(g(x,\chi)-c\big).\chi\Big), k=1,\ldots,m.\end{aligned} \tag{14}$$

Andrieu et al. (2007) and Kosuch & Lisser (2011) proposed various functions that may be chosen for h. They computed for each function a reference value for the mean square error of the obtained approximated gradient and compared them. It turned out that, the function :

$$h(x) = \frac{3}{4}\left(1-x^2\right)\left(\mathbf{1}_{\mathbb{R}^+}\right) \tag{15}$$

offered the smallest of this value Andrieu et al. (2007).

Here the indicator function $(\mathbf{1}_1)$ is defined as:

$$(\mathbf{1}_1) = \begin{cases} 1 & \text{if } 1 \leq x \leq 1,\\ 0 & \text{otherwise.}\end{cases} \tag{16}$$

Based on the results of Kosuch & Lisser (2011), we get the following estimation of the gradient of $\mathbf{J}$:

$$\begin{aligned}\nabla(\mathbf{J}_t^{(k)})(x,\chi) = \Big[(r_1^{(k)}\chi_1,\ldots,r_n^{(k)}\chi_n)^T + \Big(\sum_{\substack{j=1\\ j\neq 1}}^{n} r_{1j}^{(k)}(\chi_1+\chi_j)x_j,\ldots,\sum_{\substack{j=1\\ j\neq n}}^{n} r_{nj}^{(k)}(\chi_n+\chi_j)x_j\Big)^T\Big]\\ -d.\Big(-\frac{3}{4t}\Big(1-\Big(\frac{g(x,\chi)-c}{t}\Big)^2\Big)\cdot\mathbf{1}_1\cdot\Big(\frac{g(x,\chi)-c}{t}\Big).\chi.\big(g(x,\chi)-c\big) - \mathbf{1}_{\mathbb{R}^+}\big(g(x,\chi)-c\big).\chi\Big),\\ k=1,\ldots,m.\end{aligned} \tag{17}$$

2.3 Deterministic Equivalent Problem

The new random variable is defined as follows:

$$X := \sum_{i=1}^{n} x_i \chi_i, \tag{18}$$

which is normally distributed with mean

$$\widehat{\mu}_i := \sum_{i=1}^{n} \mu_i x_i, \tag{19}$$

standard deviation

$$\widehat{\sigma} := \sqrt{\sum_{i=1}^{n} \sigma_i^2 x_i^2}, \tag{20}$$

density function

$$\varphi(x) = \frac{1}{\widehat{\sigma}} f\left(\frac{x-\widehat{\mu}}{\widehat{\sigma}}\right) \tag{21}$$

and cumulative distribution function

$$\Phi(x) = F\left(\frac{x-\widehat{\mu}}{\widehat{\sigma}}\right). \tag{22}$$

Cohn & Barnhart (1998) and Kosuch & Lisser (2009) proposed to rewrite the objective function in a deterministic way by using the following calculations:

$$\begin{aligned}
\mathbf{E}\left[[X-c]^+\right] &= \int_{-\infty}^{\infty} [X-c]^+ \varphi(X)\, dX \\
&= \int_{c}^{\infty} (X-c)\, \varphi(X)\, dX \\
&= \int_{c}^{\infty} X \varphi(X)\, dX - c \int_{c}^{\infty} \varphi(X)\, dX \\
&= \widehat{\mu} \int_{c}^{\infty} \varphi(X)\, dX - \widehat{\sigma}^2 \int_{c}^{\infty} \varphi'(X)\, dX - c \int_{c}^{\infty} \varphi(X)\, dX \\
&= \widehat{\sigma}^2 \varphi\left[(X)\right]_c^{\infty} + (\widehat{\mu}-c)\left[\Phi(x)\right]_c^{\infty} \\
&= \widehat{\sigma}^2 \varphi(c) + (\widehat{\mu}-c)\left[1-\Phi(c)\right] \\
' &= \widehat{\sigma} f\left(\tfrac{c-\widehat{\mu}}{\widehat{\sigma}}\right) + (\widehat{\mu}-c)\cdot\left[1-F\left(\tfrac{c-\widehat{\mu}}{\widehat{\sigma}}\right)\right]
\end{aligned} \tag{23}$$

Finally, we write the deterministic equivalent objective function as follows:

$$\mathbf{J}_{det}^{k}(x) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} r_{ij}^{(k)} x_i x_j (\mu_i + \mu_j) + \sum_{i=1}^{n} r_i^{(k)} \mu_i x_i - d\left[\widehat{\sigma} f\left(\frac{c-\widehat{\mu}}{\widehat{\sigma}}\right) - (c-\widehat{\mu})\left[1-F\left(\frac{c-\widehat{\mu}}{\widehat{\sigma}}\right)\right]\right], \tag{24}$$
$$k = 1, \ldots, m.$$

3 RESOLUTION METHOD

In the decomposition framework, the original MO-SQKP is first decomposed into many Single-Objective Stochastic Quadratic Knapsack Problems (SO-SQKP). To be more precise, given the objective vector

$$F(x) = \left(f^1(x), f^2(x), \ldots, f^m(x)\right)^T, \tag{25}$$

we include the capacity constraint in the objective function by using the penalty function $[.]^+$ 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.

$$\begin{aligned} \mathbf{J}^k(x,\chi) = \quad & \mathbf{E}\Big[\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} r_{ij}^{(k)} x_i x_j(\chi_i+\chi_j)\Big] + \mathbf{E}\Big[\sum_{i=1}^{n} r_i^{(k)} x_i \chi_i\Big] - d.\mathbf{E}\Big[\sum_{i=1}^{n} x_i\chi_i - c\Big]^+, \\ & k = 1,\ldots,m. \end{aligned} \tag{26}$$

3.1 Population initialization

The initial population is constructed according to the Greedy Constructive Heuristic. Initially, the knapsack is empty. The heuristic includes the object with the highest value density relative to the remaining available objects one by one to the knapsack. The Greedy Constructive Heuristic for MO-SQKP examines relative value density $v_d(i,S)$ of an object i to select those to assign to the knapsack. The value density $v_d(i,S)$ of an object i relative to a set of other objects is the sum of the value of object i and its joint values with the objects in S divided by its weight Hiley & Julstrom (2006):

$$v_d(i,S) = \frac{\left(r_i + \sum_{j\in S} r_{ij}\right)}{w_i} \tag{27}$$

Greedy Constructive Heuristic

Input: r^*: Quadratic reward.
r: linear reward.
w: weights of objects.
C: Capacity of knapsack.
Output: A greedy solution
S $\longleftarrow$ all objects
knapsack $\longleftarrow \emptyset$
while $S \neq \emptyset$ **do**
  $b \longleftarrow$ object with maximum $v_d(b,S)$
  if $w_b \leq C$ **then**
    add b to knapsack;
    $C = C - w_b$;
  end if
  remove b from S;
end while

Subpopulation construction First, we need to manipulate our reward par objective function to create a new individual in the population. To do that, we need a *vector_reward V* at each iteration. This *vector_reward V* represents the ratio of how we want to manipulate and favorite the objective function. For instance, if we want to favorite only the first objective function, then the vector_reward look like $V = (1,0,0,...,0)$ where each value in V will multiply by the reward, i.e.

$$new\_rewards^m = V(m) * reward^m, \text{ where } k = 1,\ldots,number\ of\ objectives. \tag{28}$$

For the first *number_of_objectives* population, we want to favorite the m^{th} objective function by giving 1 to m^{th} element of V and 0 elsewhere. Then we send this *new_rewards* to our greedy algorithm in order to get a new individual. For the rest of the population, first we evaluate the previous individuals that are already in our population, then we search for the maximum value $\max^*$ from all the values evaluated. We define a new *vector_*max by

$$vector\_\max = \Big(\frac{\max(f_1^*)}{\max^*}, \frac{\max(f_2^*)}{\max^*}, ..., \frac{\max(f_m^*)}{\max^*}\Big), \tag{29}$$

where f_i^* is a vector of evaluation of the i^{th} objective across all the individuals. Based on *vector_*max we create a new *vector_reward*:

$$V = \frac{vector\_\max}{\sum vector\_\max}. \tag{30}$$

We use V as above to create a *new_rewards* and send it to our greedy algorithm in order to get a new individual. This process is repeated until we generate a population of size N.

3.2 NSGA-II Algorithm

Our resolution method is inspired by the non-dominated sorting genetic algorithm II (NSGA-II). NSGA-II is one of the most popular multi-objective optimization algorithms, it is more efficient than its previous version NSGA and tends to spread quickly. The main advantage is the strategy of preserving the diversity of solutions. NSGA-II can be detailed as follows:

Step 1. Population initialization: The population is initialized based on the problem range and constraints.

Step 2. Non dominated sort: The initialized population is sorted based on non-domination. The fast sort algorithm is described in Section 3.3.

Step 3. Crowding distance: The Crowding distance algorithm is described in Section 3.4.

Step 4. Selection: The selection of individuals is carried out using a binary tournament selection with the crowded-comparison operator.

Step 5. Genetic Operators: Real coded GA using simulated binary crossover and polynomial mutation.

Step 6. Recombination and selection: The offspring population and the current generation population are combined, and the individuals of the next generation are set by selection. The new gener-

Subpopulation construction

Input: N: Population size.
r_q: Quadratic rewards.
r: Linear rewards.
w: Weights of objects.
C: Capacity of knapsack.
Output: P: Population of size N.
$P \longleftarrow \emptyset$
for $i = 1$ to $number\_of\_objectives$ **do**
    $V = Zeros(1, number\_of\_objectives)$
    V(i)=1
    for $j = 1$ to $number\_of\_objectives$ **do**
        $new\_rewards_q^j = V(j) * r_q^j$
        $new\_rewards^j = V(j) * r^j$
    end for
    $X \longleftarrow$ Greedy Construction($new\_rewards_q$, $new\_rewards$,w,C)
    $P \longleftarrow P \cup X$
end for
while $|P| \leq N$ **do**
    max*=maximum value obtained from evaluation of P
    for $j = 1$ to $number\_of\_objectives$ **do**
        $vector\_\max(j) =$ maximum value of f^j across all individuals in P
    end for
    $V = \frac{vector\_\max}{\sum vector\_\max}$
    for $j = 1$ to $number\_of\_objectives$ **do**
        $new\_rewards_q^j = V(j) * r_q^j$
        $new\_rewards^j = V(j) * r^j$
    end for
    $X \longleftarrow$ Greedy Construction($new\_rewards_q$, $new\_rewards$,w,C)
    $P \longleftarrow P \cup X$
end while

ation is filled by each front subsequently until the population size exceeds the current population size.

3.3 Non-dominated Sort Algorithm

In this section, we develop the two steps of the *Nondominated Sorting Algorithm* whose main objective is to sort all individuals in the population according to different levels of non-dominance.

The first step aims at finding the first level of non-dominated solutions in the population of size N. This requires comparisons of each solution, for each objective function, with all the other solutions in the population to check if it is dominated. The above process (Lines 3-17 of

Algorithm 1) is repeated until all non-dominated members of the first front in the population are found. When all individuals on the first non-dominated front are found, the process proceeds to the next step to find all individuals on each subsequent non-dominated front.

Let p be any solution and denote by :

- η_p: an indicator that counts the number of solutions that dominate p.
- S_p : the set of solutions that p dominates.

After the first step of the non-dominated sorting procedure, the first non-dominated level is found and the two entities are computed for each solution. Then, all the solutions of the first non-dominated level will have their dominance number set to zero $\eta_p = 0$.

Then, in the second step of our procedure, we reduce the dominance number by one for each member q visited by each solution p with $\eta_p = 0$ of sets S_p. After that, we add, to the separate list Q, any member q if its dominance count becomes zero $\eta_q = 0$. These members belong to the second non-dominated level. We continue the above procedure with each member of Q and identify the third front. This process continues until all levels are identified. At this point, the solution is assigned a non-dominated level and will never be visited again.

3.4 Crowding-Distance Algorithm

The crowding distance value of a solution provides an estimate of the density of solutions surrounding a particular solution in the population. We calculate the average distance of two points on either side of this point along each of the objectives. Figure 1 shows the computation of the crowding distance of the i^{th} solution, which is an estimate of the size of the largest cuboid enclosing i without including any other point.

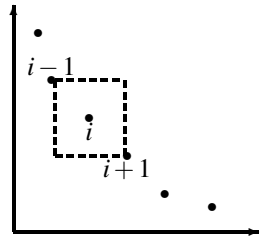

Figure 1 – Computation of the Crowding distance.

The crowding-distance computation requires sorting the population according to each objective function value in non-decreasing order. The crowding distance value of a particular solution is the average distance of its two neighboring solutions. Then, for each objective function, the boundary solutions with the lowest and the highest objective function values are given an infinite crowding distance value. All other intermediate solutions are assigned a distance value equal to the absolute normalized difference in the function values of two adjacent solutions. This process

Algorithm 1 Non-dominated Sort Algorithm

Input: N: Size of the Population;
        P: initial Population
for each $p \in P$ **do**
    $S_p = \emptyset$
    $\eta_p = 0$
    for each $q \in P$ **do**
        if $p \succ q$ **then** ▷ if p dominates q
            $S_p = S_p \cup \{q\}$ ▷ Add q to the set of solutions dominated by p
        else if $q \succ p$ **then**
            $\eta_p = \eta_p + 1$ ▷ Increment the domination counter of p
        end if
    end for
    if $\eta_p = 0$ **then** ▷ p belongs to the first front
        $p_{rank} = 1$
        $F^1 = F^1 \cup \{p\}$
    end if
end for
$i = 1$ ▷ Initialize the front counter
while $F_i \neq \emptyset$ **do**
    $Q = \emptyset$ ▷ Used to store the members of the next front
    for each $p \in F_i$ **do**
        for each $q \in S_p$ **do**
            $\eta_q = \eta_q - 1$
            if $\eta_q = 0$ **then** ▷ q belongs to the next front
                $q_{rank} = i + 1$
                $Q = Q \cup \{q\}$
            end if
        end for
    end for
    $i = i + 1$
    $F_i = Q$
end while

is done for each objective function. The overall crowding-distance value solution is computed by adding the entire individual crowding-distance value corresponding to each objective function.

Algorithm 2 below outlines the Crowding-Distance Computation procedure of all solutions in a non-dominated set P.

Algorithm 2 Crowding-Distance Computation Algorithm

Input. m: number of objective functions;
    P: initial Population
$\mathscr{L} = |P|$ ▷ Number of solutions in P
for each i **do**
    $P[i]_{distance} = 0$ ▷ Initialize distance
end for
for each objective function m **do**
    $P = Sort(P,m)$ ▷ Sort using each objective value
    $P[1]_{distance} = P[\mathscr{L}]_{distance} = \infty$ ▷ Boundary points are always selected
    for $i = 2$ **to** $\mathscr{L} - 1$ **do** ▷ For all other points
        $$P[i]_{distance} = P[i]_{distance} + \frac{P[i+1].m - P[i-1].m}{f^m_{\max} - f^m_{\min}}$$
    end for
end for

Note that $P[i].k$ refers to the k^{th} objective function value of the individual i in the set P and $f^k_{\max}$ and $f^k_{\min}$ are respectively the maximum and the minimum values of the k^{th} objective function.

At this stage, all population members in the set P are assigned a distance metric. A solution with a smaller value of this distance measure is, in some sense, more crowded by other solutions. This is exactly what we compare in the proposed Crowded-comparison operator above.

Crowded-comparison operator The crowding comparison operator guides the selection process at different stages of the algorithm towards a uniformly spread Pareto optimal front. The crowding distance is introduced only when it is a must to select individuals of the same non-domination rank, i.e., the crowding distance is a criterion for selecting individuals of the same rank.

Suppose that each individual i in the population has two attributes:

- non-domination rank (i_{rank}),
- crowding distance $(i_{distance})$.

The individuals are selected by using a binary tournament selection with crowded comparison-operator.

- If $(x_{rank}) < (y_{rank})$.
- If x and y belong to a same front $\mathscr{F}_i$ and $(x_{distance}) > (y_{distance})$.
- If x and y are both solutions and are in different ranks $(x_{rank}) \neq (y_{rank})$, a solution with lower rank is selected.

- If x and y are both solutions and are in different ranks $(x_{rank}) = (y_{rank})$, a solution that is located in a higher crowded region is selected.
- If x and $y$are both solutions and are in a same rank then a solution from greater crowded space will be selected.

3.5 Selection Neighborhood Pareto Local Search (SNPLS) Algorithm

Algorithm 3 Selection Neighborhood Pareto Local Search SNPLS Algorithm

Input: m, the number of objectives;
Q, the set of new offspring.
Output: Procedure to compute one-step Local Search to improve I_{old} to I_{new}.
for $k = 1$ **to** m **do**
    Compute ω_k with respect to I_{old};
    Compute adaptive weight λ^k with respect to I_{old};
end for
Construct Single objective function $\mathbf{J}(x, \lambda)$;
$I_{new} \leftarrow$ Gradient algorithm $\mathbf{J}(x, \lambda)$ of I_{old};
Evaluate Fitness (I_{new});
if I_{new} is better than I_{old} **then**
    return I_{new};
else return I_{old};
end if

In the decomposition framework, the original MO-SQKP is first decomposed into many SO-SQKP. To be specific, given the objective vector $F(x) = \left(f^1(x), f^2(x), \ldots, f^m(x)\right)^T$ and weight vector $\lambda = (\lambda^1, \ldots, \lambda^m)^T$, where the sum of weights vector should be equal to 1, the objective function of a subproblem is stated as:

$$\begin{cases} \mathbf{J}(x, \chi) = \sum_{k=1}^{m} \lambda^{(k)} f^{(k)}(x) & (31.1) \\ \sum_{k=1}^{m} \lambda^{(k)} = 1 & (31.2) \\ \lambda^{(k)} \geq 0, \quad k = 1, \ldots, m & (31.3) \end{cases} \tag{31}$$

We can write $J(x, \chi)$ as follows:

$$\mathbf{J}^{(k)}(x, \chi) = \sum_{k=1}^{m} \lambda^{(k)} . \left[\mathbf{E}\Big[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} r_{ij}^{(k)} x_i x_j (\chi_i + \chi_j) \Big] + \mathbf{E}\Big[\sum_{i=1}^{n} r_i^{(k)} x_i \chi_i \Big] - d\mathbf{E}\Big[\sum_{i=1}^{n} x_i \chi_i - c \Big]^+ \right], \tag{32}$$

and an approximation of the gradient of the function $\mathbf{J}(x,\chi)$ is given by:

$$\nabla(\mathbf{J}_t^{(k)})(x,\chi) = \lambda^{(k)}\left[\left[(r_1^{(k)}\chi_1,\ldots,r_n^{(k)}\chi_n)^T + \Big(\sum_{\substack{j=1\\ j\neq 1}}^{n} r_{1j}^{(k)}(\chi_1+\chi_j)x_j,\ldots,\sum_{\substack{j=1\\ j\neq n}}^{n} r_{nj}^{(k)}(\chi_n+\chi_j)x_j\Big)^T\right]\right.$$
$$\left. - d.\Big(-\frac{3}{4t}\Big(1-\Big(\frac{g(x,\chi)-c}{t}\Big)^2\Big)\cdot\mathbf{1}_1\cdot\Big(\frac{g(x,\chi)-c}{t}\Big).\chi.\big(g(x,\chi)-c\big) - \mathbf{1}_{\mathbb{R}^+}\big(g(x,\chi)-c\big).\chi\Big)\right], \tag{33}$$
$$k = 1,\ldots,m.$$

The weight vectors are calculated based on the value of the Euclidean norm and parameters ω_i as follows:

$$\lambda^k = \frac{\omega^k}{\sum_{i=1}^{m}\omega^i}, \tag{34}$$

where:

- ω_k is given by:

$$\omega^k = \frac{f^k(x)}{\|f(x)\|}, \tag{35}$$

- m is the number of objective functions.
- $f^k(x)$ is the value of the k^{th} objective function of a solution x,
- $\|f(x)\|$ is the Euclidean norm, given by:

$$\|f(x)\| = \sqrt{(f^1(x))^2+\ldots+(f^m(x))^2} \tag{36}$$

Once the individual weights are determined for all the objectives, they are combined together into a single objective F as follows:

$$F = \lambda^1 f^1 + \lambda^2 f^2 + \ldots + \lambda^m f^m \tag{37}$$

We then apply local gradient algorithm to compute the new local solution x^{p+1} from the current solution x^p.

After that, the Evaluate Fitness is done to new solution because the value of fitness is our parameter of comparison between the new and the old solutions. Finally, we choose a solution that has the best fitness.

We repeat the above process for each offspring.

After having built these twice Algorithms (Non-dominated Sort Algorithm,Crowding-Distance Computation Algorithm) including a *crowded comparison operator* and determining different parameters (i_{rank}) and $(i_{distance})$, we are now ready to describe the Memetic algorithm.

Algorithm 4 Gradient algorithm

Choose $x^0 \in \mathbf{X}_{ad} = [0,1]^n$
At step p, $X^p = (X_1^p, \ldots, X_n^p)$ of χ according to its normal distribution
Update x^p as follows: $x^{p+1} = x^p + \varepsilon^p r^p$ ▷ where $r^p = \nabla_x \mathbf{J}(x^p, \chi^p)$ and $(\varepsilon^p)_{p\in\mathbb{N}}$ is the step size.
for $i = 1$ **to** n **do**
    if $x_i^{p+1} \succ 1$ **then**
        Set $x_i^{p+1} = 1$
        if $x_i^{p+1} \prec 0$ **then**
            Set $x_i^{p+1} = 0$
        end if
    end if
end for

3.6 Memetic Algorithm with Selection Neighborhood Pareto Local Search

In this section, we detail our Memetic Algorithm with Selection Neighborhood Pareto Local Search Algorithm MASNPL for MO-SQKP, including initialization, crossover, and local search.

Then, the fitness of each individual is evaluated and Non-dominated Sorting is applied to assign a non-domination rank i_{rank} equal to its non-domination level, and Crowding-Distance is computed for each individual i in the population $i_{distance}$.

These two parameters, i_{rank} and $i_{distance}$, are used to select individuals in the most crowded region and maintain the diversity of solutions on the Pareto front. Then, binary tournament selection is applied to choose the parents, after which, crossover and mutation operators are performed to generate new candidate solutions, i.e., the offspring population of size N.

Crossover operator Select randomly two parents $(chromosomes)$ $P^1 = \{x_1^1, x_2^1, x_3^1, \ldots, x_n^1\}$ and $P^2 = \{x_1^2, x_2^2, x_3^2, \ldots, x_n^2\}$ from the current population. A random crossover point is selected. Hence, two substrings are generated before and after cut point in each of the parent $(chromosomes)$. At this cut, the genetic information to the left (or right) of the point is swapped between the two parents $(chromosomes)$ to produce two offspring chromosomes $(children)$.

Copy first substring from P^1 and copy the second substring from P^2 and insert as they are in offspring 1.

Copy first substring from P^2 and copy the second substring from P^1 and insert as they are in offspring 2.

Algorithm 5 Memetic Algorithm With Selection Neighborhood Pareto Local Search (MASNPL)

Input: P: the current Population of size N;
    N: the size of the population;
    Q: the set of new offspring;
    N: the size of new offspring.
Initialize the population to size N;
Evaluate Fitness for every individual in the population of size N;
Apply Non-dominated Sort Algorithm (P,N);
Compute Crowding-Distance Computation Algorithm (P,N);
$NG = 1$;
while $NG \leq NG_{\max}$ **do**
    $Q = \emptyset$;
    for $k = 1$ **to** $\frac{P}{2}$ **do**
        Select Parents using Binary Tournament;
        Apply the crossover operator to generate two new offsprings (Q_1, Q_2);
        Apply Mutation operator on both (Q_1, Q_2) with probability $P_m = \frac{1}{n}$;
        for $j = 1$ **to** 2 **do**
            $Q_j \longleftarrow$ the Selection Neighborhood Pareto Local Search (Q_j);
            $Q = Q \cup \{Q_j\}$;
        end for
    end for
    $R_t = P_t \cup Q_t$;
    Apply Non-dominated Sort Algorithm (R_t)**;**
    $P_{t+1} = \emptyset$ and $i = 1$;
    while $(|P_{t+1}| + |_i|) \leq N$ **do** ▷ until the parent population is filled
        Crowding-Distance Computation (F_i); ▷ calculate crowding-distance in F_i
        $P_{t+1} = P_{t+1} \cup F_i$; ▷ include i-th non-dominated front in the parent population
        $i = i + 1$ ▷ check the next front for inclusion
    end while
    Sort F_i; ▷ sort in descending order using crowded comparison operator
    $P_{t+1} = P_{t+1} \cup F_i[1, N - |P_{t+1}|]$; ▷ choose the first $(N - |P_{t+1}|)$ elements of F_i
    $Q_{t+1} =$ Make New Population (P_{t+1}); ▷ use selection, crossover and mutation to create a new population (Q_{t+1})
    $NG = NG + 1$;
end while

Mutation Mutation is an operator that applies changes randomly to one or more genes to produce and introduce the diversity to a new offspring and is usually applied with a low probability p_m.

In the mutation stage, one individual is selected randomly from the current population, then it is flipped from 0 to 1 or from 1 to 0. After that, this individual is inserted in the population. The mutation probability is fixed to 0.95.

Tournament We select two individuals randomly from the current population. After that, there is a competition amongst the selected individuals. The competition is used to determine the individual with the highest fitness value to be used in the generation of the new population. The individual winner has the best non-domination rank.

The next stage is to apply a single step of the SNPLS Algorithm on the new offspring. The combined population $(R_t = P_t \cup Q_t)$ is formed where:

- P_t : is the parent population of size N.
- Q_t : is the offspring population of size N.
- R_t : is the combined population of size $2N$.

Non-dominated sorting and Crowding distance are applied to a combined population. Then, the population R_t is sorted according to non-domination. Now, solutions belonging to the best non-dominated set $\mathscr{F}_1$ are of the best solutions in the combined population and must be emphasized more than any other solution in the combined population.

If the size of $\mathscr{F}_1$ equals N, we definitely add all members of the set $\mathscr{F}_1$ to the new population P_{t+1}.

If the size of $\mathscr{F}_1$ is smaller than N, we definitely put all members of the set $\mathscr{F}_1$ in the new population P_{t+1}. The remaining members of the population P_{t+1} are chosen from subsequent non-dominated fronts in the order of their ranking.

Thus, solutions from the set $\mathscr{F}_2$ are chosen next, followed by solutions from the set $\mathscr{F}_3$, and so on.

The above process (Lines 17-20 of Algorithm 4) is continued until a final set of non-dominated solutions of size N is obtained. The new population P_{t+1} of size N is now used for selection, binary tournament selection operator, crossover, and mutation to create a new population Q_{t+1} of size N. The above process (Lines 6-23 of Algorithm 4) is repeated until $NG_{\max}$ is obtained.

4 NUMERICAL RESULTS

In this section, we present our numerical results. As the multi-objective stochastic quadratic knapsack problem is not treated before, and in order to evaluate the performance of our resolution method, we suggest comparing our algorithms firstly with an exact method, and secondly with NSGA-II. The results of our comparison are presented in the following subsections.

Note that all our experiments are realized on an Intel core i5-4200U CPU machine with 1.60 Ghz and 4 Go of RAM.

4.1 Performance comparison of MASNPL and an exact method

To solve the multi-objective stochastic knapsack problem with simple recourse by an exact method, we implement the equivalent deterministic problem already found and execute it on randomly generated instances. Note that the comparison is performed for three examples, and each algorithm is executed once. The results for each example are shown in the figures below.

The instances are created randomly with the parameters given below.

Example 4.1.1.

- *Number of objective functions:* 2.
- *Weights : uniformly distributed with mean* 225 *and variance* 25.
- *Rewards per weight: generated uniformly between* 1 *and* 10.
- *Number of objects is* 50.
- *Knapsack capacity is* 9000.
- *Penalty d; equal to* 330.
- *Number of generations for MASNP:* 50.

The result of our comparison is presented graphically on Figure 2.

The figure shows the non-dominated solutions obtained by an exact method and our resolution method (MASNPL algorithm). The CPU-time needed to obtain the final solution with an exact method is: 52179,75 seconds ($\simeq$ 15 hours and 30 mn) and the total CPU-time with the MASNPL algorithm is 41,62 seconds (less than a minute). MASNPL algorithm is, of course, faster than the exact method and provides good quality solutions.

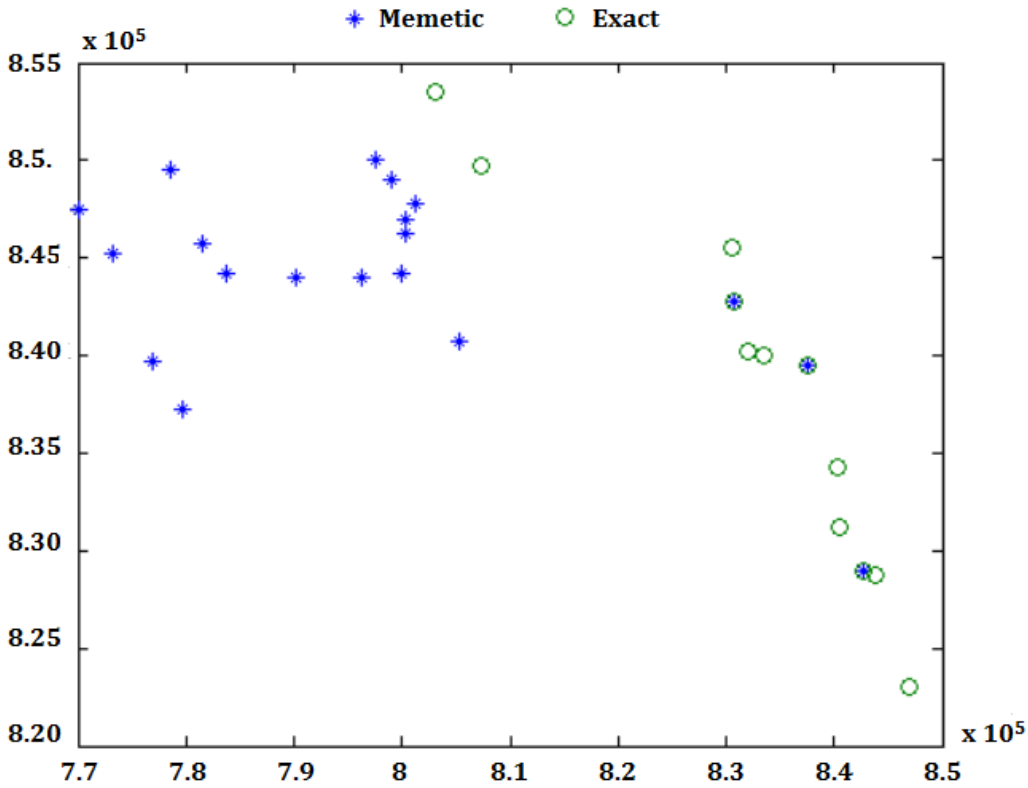

Figure 2 – Comparison between Pareto Fronts of an exact algorithm and MASNPLfor 50 objects.

Example 4.1.2.

- *Number of objective functions:* 2.
- *Weights : uniformly distributed with mean* 225 *and variance* 25.
- *Rewards per weight: generated uniformly between* 1 *and* 10.
- *Number of objects is* 20.
- *Knapsack capacity is* 3600.
- *Penalty d; equal to* 132.
- *Number of generations for MASNP:* 50.

The result of our comparison is presented graphically in Figure 3.

The figure shows the non-domination solutions obtained by an exact method and our resolution method (MASNPL algorithm). The CPU-time needed to obtain the final solution with an exact

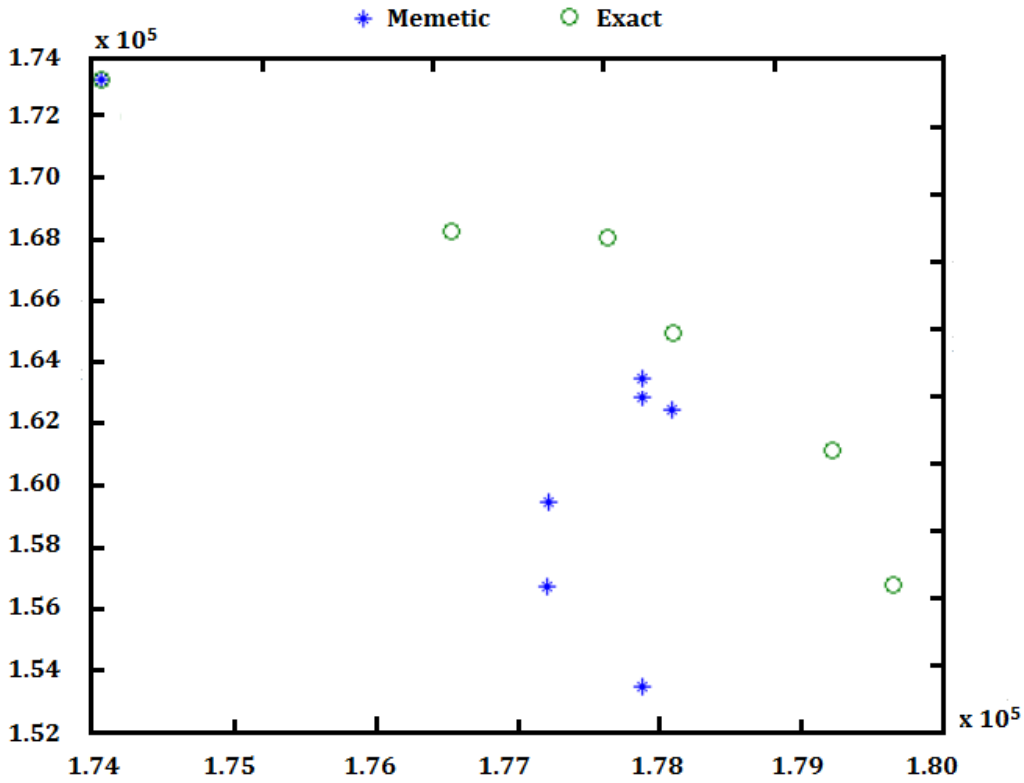

Figure 3 – Comparison between Pareto Fronts of an exact algorithm and MASNPL for 20 objects.

method is: 2983,02 seconds (≃49 minutes) and the total CPU-time with the MASNPL algorithm is 43,69 seconds (less than a minute). MASNPL algorithm is, of course, faster than the exact method and provides good quality solutions.

Example 4.1.3.

- *Number of objective functions:* 2.
- *Weights : uniformly distributed with mean* 225 *and variance* 25.
- *Rewards per weight: generated uniformly between* 1 *and* 10.
- *Number of objects is* 15.
- *Knapsack capacity is* 2700.
- *Penalty d; equal to* 99.
- *Number of generations for MASNP:* 50.

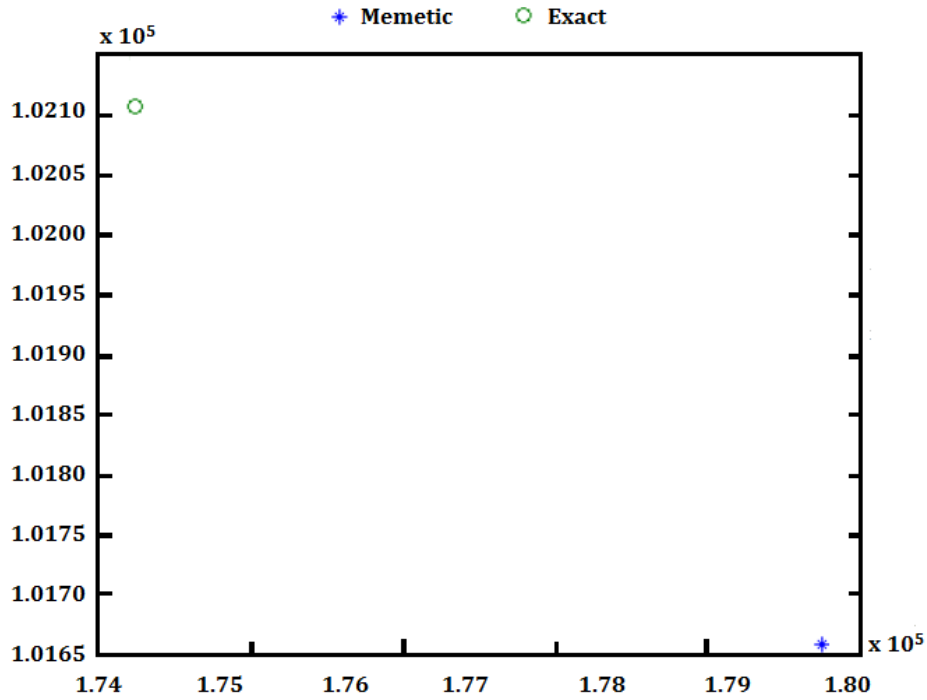

Figure 4 – Comparison between Pareto Fronts of an exact algorithm and MASNPLfor 15 objects.

The result of our comparison is presented graphically on Figure 4.

The figure shows the non-domination solutions obtained by an exact method and our resolution method (MASNPL algorithm). The CPU-time needed to obtain the final solution with an exact method is: 1175,48 seconds (≃ 20 minutes) and the total CPU-time with the MASNPL algorithm is 35,91 seconds. MASNPL algorithm is, of course, faster than the exact method and provides good quality solutions.

4.2 Performance comparison of MASNPL and NSGA-II

To compare our MASNPL algorithm and NSGA-II, we implemented these methods on MATLAB and used the multi-objective stochastic quadratic knapsack instances created randomly with the following common parameters:

- Weights : uniformly distributed with mean equal to 225 and variance equal to 25.
- Rewards per weight: generated uniformly between 1 and 10.
- Numbers of generations: 50 for MASNPL, and 2500 for NSGA-II.

We iterate each algorithm 20 times to create a final population of size 50, and in each iteration, the algorithms (MASNPL and NSGA-II) use the same parameters and the same instance created above.

We calculate the value of the objective functions f_i for each individual in the population. After this, we sort the population into different levels of non-domination, and we assign the value of the crowding distance to each individual in the population. For each iteration of the $20h$ runs, we save the non-dominated solutions (rank equal to 1).

The next step of our comparison consists to merge all the non-dominated solutions already saved and obtained by (MASNPL and NSGA-II) algorithm, then applying the non-domination sort algorithm in order to sort (rank) the solutions merged into different non-domination levels and attribute the Crowding-Distance values for each individual of the population merged by applying the crowding distance algorithm.

Example 4.2.1. *The parameters of this instance are:*

- *Number of objective functions:* 2.
- *Number of objects:* 50.
- *Knapsack capacity:* 9000.
- *Penalty d:* 330.

Table 1 represents the results (number of non-dominated solutions) obtained by MASNPL and NSGA-II for each test.

Table 1 – Numbers of non-dominated solutions obtained by MASNPL and NSGA-II.

	Number of solutions of Rank 1		CPU-time (s)		Ratio %	
	MASNPL	NSGA-II	MASNPL	NSGA-II	MASNPL	NSGA-II
1	10	0	1713,51311	3419,76764	100	0
2	6	0	1609,18844	3376,87798	100	0
3	10	0	1516,1261	3250,44791	100	0
4	20	5	1347,10586	2879,00458	80	20
5	2	0	1400,02226	2831,42838	100	0
6	5	0	1319,84907	2909,47836	100	0
7	5	0	1328,41828	2930,77582	100	0
8	6	1	1418,68944	2882,2089	85,71	14,28
9	5	1	1357,27806	2911,37711	83,33	16,67
10	16	2	1422,045	2757,8156	88,89	11,11

To evaluate the quality of the solutions obtained by applying the MASNPL and NSGA-II algorithms, we use the ratio that determines the percentage of the number of non-dominated solutions for each algorithm compared to the total number of 1-rank solutions.

For the 10 tests, we note that for 6 tests, the 1 rank solutions obtained by the MASNPL algorithm dominate the solutions obtained by the NSGA-II algorithm (the number of 1 rank solutions is equal to 0) although the generation number for NSGA-II is 50 times the generation number for the MASNPL algorithm.

For the rest of the tests, we can observe that:

- the number of non-dominated solutions obtained by the MASNPL algorithm is greater than the number of non-dominated solutions obtained by the NSGA-II algorithm,
- the 1 rank solutions obtained by the MASNPL algorithm are more crowded than the solutions obtained by NSGA-II, i.e., they are located in a more crowded region,
- in terms of CPU time, MASNPL is faster than NSGA-II.

According to these results, we can conclude that the MASNPL algorithm performs significantly better than the NSGA-II algorithm.

Example 4.2.2. *The parameters of this instance are:*

- *Number of objective functions:* 2.
- *Number of objects:* 100.
- *Knapsack capacity:* 21000.
- *Penalty d:* 7520.

Table 2: represents the results (number of the non-domination solutions) obtained by the (MASNPL and NSGA-II) algorithm for every test.

Table 2 – Numbers of non-dominated solutions obtained by MASNPL and NSGA-II.

	Number of solutions of Rank 1		CPU-time (s)		Ratio %	
	(MASNPL)	NSGA-II	MASNPL	NSGA-II	MASNPL	NSGA-II
1	4	0	2555,20082	4160,13939	100	0
2	1	1	2485,20063	4235,54793	50	50
3	3	0	2393,36046	4176,23395	100	0
4	8	0	2481,62	3965,26625	100	0
5	1	0	2501,27197	4039,40369	100	0
6	1	0	2578,42903	4144,85823	100	0
7	8	0	2270,357	4106,02298	100	0
8	1	0	2590,46484	8054,85057	100	0
9	1	0	2504,63287	4205,24089	100	0
10	16	0	2092,41251	3490,4851	100	0

In Table 2, for the 10 tests, we note that in 09 tests, the 1 rank solutions obtained by MASNPL dominate the solutions obtained by NSGA-II (no 1 rank solution), although the generation number for NSGA-II is 50 times the generation number for the MASNPL algorithm.

In the remaining test, the number of non-dominated solutions obtained by MASNPL equals the number of 1 rank solutions obtained by NSGA-II.

In terms of CPU time, for all tests, MASNPL is faster than NSGA-II.

According to these results, we can say that the MASNPL algorithm performs significantly better than NSGA-II.

Example 4.2.3. *The parameters of this instance are:*

- *Number of objective functions:* 2.
- *Number of objects:* 200.
- *Knapsack capacity:* 40000.
- *Penalty d:* 1424.

Table 3: represents the results (number of the non-domination solutions) obtained by the (MASNPL and NSGA-II) algorithm for every test.

Table 3 – Numbers of non-dominated solutions obtained by MASNPL and NSGA-II.

	Number of solutions of Rank 1		CPU-time (s)		Ratio %	
	(MASNPL)	NSGA-II	MASNPL	NSGA-II	MASNPL	NSGA-II
1	5	0	4298,38137	4627,86451	100	0
2	5	0	4329,33696	4577,69908	100	0
3	2	1	4286,61502	4587,7257	66,67	33,33
4	1	0	4457,91234	4567,6765	100	0
5	5	0	4040,51267	4342,15335	100	0
6	1	0	4341,55699	4465,75378	100	0
7	3	0	4451,06075	4527,91025	100	0
8	3	0	4537,64983	4554,9791	100	0
9	5	0	4267,92233	4471,04282	100	0
10	1	0	4479,37789	4662,25679	100	0

In Table 3, for the 10 tests, we note that in 09 tests, the 1 rank solutions obtained by MASNPL dominate the solutions obtained by NSGA-II (no 1 rank solution), although the generation number for NSGA-II is 50 times the generation number for the MASNPL algorithm.

In the remaining test, the number of non-dominated solutions obtained by MASNPL is twice the number of 1 rank solutions obtained by NSGA-II.

In terms of CPU time, for all tests, MASNPL is faster than NSGA-II.

According to these results, we can say that the MASNPL algorithm performs significantly better than NSGA-II.

Example 4.2.4. *The parameters of this instance are:*

- *Number of objective functions:* 4.
- *Number of objects:* 50.
- *Knapsack capacity:* 9000.
- *Penalty d:* 330.

Table 4: represents the results (number of the non-domination solutions) obtained by the (MASNPL and NSGA-II) algorithm for every test.

Table 4 – Numbers of non-dominated solutions obtained by MASNPL and NSGA-II.

	Number of solutions of Rank 1		CPU-time (s)		Ratio %	
	(MASNPL)	NSGA-II	MASNPL	NSGA-II	MASNPL	NSGA-II
1	66	26	1851,74066	2445,45638	71,74	28,26
2	31	2	1905,95985	2648,07941	93,94	6,06
3	71	22	1905,95985	2648,07941	76,34	23,65
4	69	19	1930,86798	2560,80004	78,41	21,59
5	29	9	1886,09592	2627,31683	76,31	23,68
6	14	31	1951,97476	2582,64445	36,84	68,69
7	97	10	1946,74633	2408,61901	90,65	9,34
8	13	5	1938,47225	2592,48961	72,22	27,78
9	1	13	2003,46515	2616,81209	7,14	92,86
10	9	11	1965,69301	2734,79297	45	55

In Table 4, for the 10 tests, we note that in 7 tests, MASNPL obtained more solutions of rank 1 than NSGA-II, although the generation number for NSGA-II is 50 times the generation number for MASNPL. We also observe that the non-dominated solutions obtained by MASNPL are more crowded (i.e., they are located in a greater crowded region) than the non-dominated solutions obtained by NSGA-II.

In the remaining (3) tests, the number of non-dominated solutions obtained by MASNPL is less than the number of the solutions of rank 1 obtained by NSGA-II, but the solutions of rank 1, obtained by MASNPL, are more crowded than those obtained by NSGA-II.

In terms of CPU time, for all tests, MASNPL is faster than NSGA-II.

According to these results, we can conclude that the MASNPL algorithm performs significantly better than NSGA-II.

Example 4.2.5. *The parameters of this instance are:*

- *Number of objective functions:* 3.
- *Number of objects:* 100.
- *Knapsack capacity:* 21000.
- *Penalty d:* 752.

Table 5 represents the results (number of the non-domination solutions) obtained by the (MASNPL and NSGA-II) algorithm for every test.

Table 5 – Numbers of non-dominated solutions obtained by MASNPL and NSGA-II.

	Number of solutions of Rank 1		CPU-time (s)		Ratio %	
	(MASNPL)	NSGA-II	MASNPL	NSGA-II	MASNPL	NSGA-II
1	18	6	3338,65774	4686,83084	75	25
2	1	0	3255,72489	4753,71708	100	0
3	17	1	3316,3073	4848,68752	94,44	5,55
4	10	1	3164,97535	4761,14919	90,91	9,09
5	0	2	3261,58644	4751,8666	0	100
6	18	6	3251,30331	4100,82273	75	25
7	1	0	3204,16986	4137,24102	100	0
8	17	1	3233,53408	4234,42546	94,44	5,55
9	10	1	3138,91231	4260,57573	90,91	9,09
10	1	0	3101,46999	4057,35701	100	0

In Table 5, for the 10 tests, we note that in 3, the solutions of rank 1 obtained by MASNPL dominate those obtained by NSGAII (that did not get any 1 rank solution), although the generation number for NSGA-II is 50 times those for MASNPL. We also observe that the non-dominated solutions obtained by MASNPL are more crowded.

In 6 tests, MASNPL obtained more solutions than NSGA-II, and the non-dominated solutions obtained by MASNPL are more crowded than those obtained by NSGA-II.

But for the remaining test, the solutions of rank 1 obtained by NSGA-II dominate those obtained by MASNPL.

In terms of CPU time, for all tests, MASNPL is faster than NSGA-II.

According to these results, we can conclude that the MASNPL algorithm performs significantly better than NSGA-II.

Example 4.2.6. *The parameters of this instance are:*

- *Number of objective functions:* 5.
- *Number of objects:* 150.
- *Knapsack capacity:* 28000.
- *Penalty d:* 1008.

Table 6 represents the results (number of the non-domination solutions) obtained by the (MASNPL and NSGA-II) algorithm for every test.

Table 6 – Numbers of non-dominated solutions obtained by MASNPL and NSGA-II.

	Number of solutions of Rank 1		CPU-time (s)		Ratio %	
	(MASNPL)	NSGA-II	MASNPL	NSGA-II	MASNPL	NSGA-II
1	14	15	4994,0142	7138,69308	48,27	51,72
2	6	11	5149,28687	7544,84421	35,29	64,7
3	2	3	5273,99488	7636,07286	40	60
4	30	21	5119,07871	7084,38893	58,82	41,18
5	12	2	5262,87698	7011,5114	85,71	14,28
6	3	1	4743,11459	6919,07195	75	25
7	4	0	4885,45103	7210,53789	100	0
8	3	1	4791,12316	7187,80439	75	25
9	5	1	5134,35748	7932,27036	83,33	16,67
10	3	1	5063,31278	7366,53774	75	25

In Table 6, for the 10 tests, we note that in 1 test, MASNPL obtained solutions of rank 1, but NSGA-II did not, although the generation number for NSGA-II is 50 times those for MASNPL. We also can observe that the non-dominated solutions obtained by MASNPL are more crowded than those obtained by NSGA-II.

In 6 tests, MASNPL obtained more solutions than NSGA-II, and the non-dominated solutions obtained by MASNPL are also more crowded than those obtained by NSGA-II.

However, for the remaining 3 tests, the solutions of rank 1 obtained by NSGA-II dominate those obtained by MASNPL.

In terms of CPU time, for all tests, MASNPL is faster than NSGA-II.

According to these results, we can conclude that the MASNPL algorithm performs significantly better than NSGA-II.

5 CONCLUSION

In this paper, we detailed the model for the Multi-objective Stochastic Quadratic Knapsack Problem with simple recourse and random weights. As the objective functions are not differentiable,

we approximate their gradients by the approximation by the convolution method. We apply a greedy heuristic for MO-SQKP to obtain an initial population. Then we use the Non-dominated Sort Algorithm to sort the population into different non-domination levels. After that, we determine the crowding distance value of a solution by applying the Crowding-Distance Computation Algorithm. We obtain a population sorted by non-domination levels and the crowding distance for each individual. We then apply a series of mutations, crossovers, and local searches to this population to generate an offspring population. To improve the offspring population, we apply the Selection Neighborhood Pareto Local Search SNPLS algorithm based on the comparison between a current solution (offspring), and a new solution obtained by the gradient algorithm. Then, the Non-dominated Sort Algorithm and the Crowding-Distance Computation Algorithm are applied to the improved offspring to select our first final best individuals of the population. Finally, the experimental results for the comparison between MASNPL and NSGA-II show that using the gradient algorithm with NSGA-II performs significantly better and more efficiently than NSGA-II.

Our study may open new research perspectives for the stochastic quadratic multi-objective knapsack problem. The methodology we adopted can easily be adapted to two-stage or multi-stage problems where the weight or reward is not known in advance. Moreover, the efficiency of our method shows that hybridization allows gains both in terms of quality of the obtained solutions and in execution time. Other ideas could be inserted in the algorithm to improve it further.

Acknowledgements

The authors thank the anonymous referees for their helpful comments and suggestions which contributed to the improvement of the contents of this paper.

The authors wish to thank the Directorate-General of the Scientific Research and the Technological Development of Algeria for its institutional support.

References

AÏDER M, GACEM O & HIFI M. 2022a. Branch and solve strategies-based algorithm for the quadratic multiple knapsack problem. *Journal of the Operational Research Society*, **73**(2): 540–557.

AÏDER M, GACEM O & HIFI M. 2022b. A hybrid population-based algorithm for the bi-objective quadratic multiple knapsack problem. *Expert Systems With Applications*, **191**: 116238.

ANDRIEU L, COHEN G & VÁZQUEZ-ABAD F. 2007. Stochastic programming with probability constraints. *Arxiv*, Available at: http://fr.arxiv.org/abs/0708.0281.

ARSHAD S, YANG S & LI C. 2009. A sequence based genetic algorithm with local search for the travelling salesman problem. In: *Proceedings of the 2009 UK Workshop on Computational Intelligence*. pp. 98–105.

BHUVANA J & ARAVINDAN C. 2016. Memetic algorithm with Preferential Local Search using adaptive weights for multi-objective optimization problems. *Soft Comput.*, (20): 1365–1388.

BLADO D & TORIELLO A. 2021. A column and constraint generation algorithm for the dynamic knapsack problem with stochastic item sizes. *Mathematical Programming Computation*, (13): 1.

CACCHIANI V, IORI M & MARTELLO ALS. 2022a. Knapsack problems - An overview of recent advances. Part I: Single knapsack problems. *Computers & Operations Research*, **143**: 1056922.

CACCHIANI V, IORI M & MARTELLO ALS. 2022b. Knapsack problems - An overview of recent advances. Part II: Multiple, multidimensional, and quadratic knapsack problem. *Computers & Operations Research*, **143**: 1056923.

CHEN Y & HAO J. 2016. The bi-objective quadratic multiple knapsack problem: Model and heuristics. *Knowl.-Based Syst.*, pp. 89–100.

CHU X & YU X. 2018. Improved Crowding Distance for NSGA-II. *Arxiv*, Available at: https://arxiv.org/abs/1811.12667v1.

COHN A & 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)*.

DEB K, PRATAP A, AGARWAL S & MEYARIVAN T. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, **6**(2): 182–197.

GALLO G, HAMMER PL & SIMEONE B. 1980. *Quadratic knapsack problems*. Springer.

HILEY A & JULSTROM B. 2006. The quadratic multiple knapsack problem and three heuristic approaches to it. In: *Proceedings of the 8th annual conference on Genetic and evolutionary computation (GECCO'06)*. pp. 547–552.

KELLERER H, PERSCHY U & PISINGER D. 2004. *Knapsack problems*. Springer.

KIM H & LIOU MS. 2012. New fitness sharing approach for multi-objective genetic algorithms. *J Glob Optim*, **55**(3): 579–595.

KOSUCH S, LETOURNEL M & LISSER A. 2017. Stochastic Knapsack Problem: Application To Transportation Problems. *Pesquisa Operacional*, **37**(3): 597–613.

KOSUCH S & LISSER A. 2009. Upper bounds for the 0-1 stochastic knapsack problem and a B & B algorithm. *Ann. Oper. Res.*, **176**(1): 77–93.

KOSUCH S & LISSER A. 2011. On two-stage stochastic knapsack problems. *Discrete Appl. Math.*, **159**(16): 1827–1841.

LÉTOCART L, PLATEAU MC & PLATEAU G. 2014. An efficient hybrid heuristic method for the 0-1 exact k-item quadratic knapsack problem. *Pesquisa Operacional*, **34**(1): 49–72.

LISSER A & LOPEZ R. 2010. Stochastic Quadratic Knapsack with Recourse. *Electronic Notes in Discrete Mathematics*, **36**: 97–104.

MARTELLO S & TOTH P. 1990. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons Ltd.

MEI Y, TANG K & YAO X. 2011. Decomposition-Based Memetic Algorithm for Multiobjective Capacitated Arc Routing Problem. *IEEE Transactions on Evolutionary Computation*, **15**(2): 151–165.

PISINGER D. 2007. The quadratic knapsack problem-a survey. *Discrete Appl. Math.*, **155**(5): 623–648.

RANGE TM, KOZLOWSKI D & PETERSEN NC. 2018. A shortest-path-based approach for the stochastic knapsack problem with non-decreasing expected overfilling costs. *Computers & Operations Research*, **97**: 111–124.

SONG B, LI Y, CHEN Y, YAO F & CHEN Y. 2018. A Repair-based approach for stochastic quadratic multiple knapsack problem, **145**: 145–155.

TANG K, MEI Y & YAO X. 2009. Memetic Algorithm With Extended Neighborhood Search for Capacitated Arc Routing Problems. *IEEE Trans Evol Comput*, **13**(5): 1151–116.

TÖNISSEN DD & SCHLICHER L. 2021. Using 3D-printing in disaster response: The two-stage stochastic 3D-printing knapsack problem Author links open overlay panel, **133**: 105356.

TÖNISSEN DD, VAN DEN AKKER JM & HOOGEVEEN JA. 2021. Column generation strategies and decomposition approaches for the two-stage stochastic multiple knapsack problem. *Computers & Operations Research*, **83**: 125–139.

ZHANG Q & LI H. 2007. MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition. *IEEE Trans. Evol. Comput.*, **11**: 712–731.

ZHOU Q, HAO JK & WU Q. 2022. A hybrid evolutionary search for the generalized quadratic multiple knapsack problem. *European Journal of Operational Research*, **296**(3): 788–803.

How to cite

GUERROUMA A AND AÏDER M. 2022. A hybridized multi-objective memetic algorithm for the multi-objective stochastic quadratic knapsack problem. *Pesquisa Operacional*, **42**: 257386. doi: 10.1590/0101-7438.2022.042.00257386.