A cooperative local search-based algorithm for the Multiple-Scenario Max-Min Knapsack Problem

Abdelkader Sbihi

A Cooperative Local Search-Based Algorithm for the Multiple-Scenario Max-Min Knapsack Problem

Abdelkader Sbihi

Department of Information Systems and Decision Making, Audencia Nantes-School of Management, 8 route de la Jonelière 44312 Nantes Cedex 3, France
Email: asbihi@audencia.com

Abstract

The purpose of this article is to present a novel method to approximately solve the Multiple-Scenario Max-Min Knapsack Problem (MSM^{2}KP). This problem models many real world situations, e.g. when for many scenarios noted $\pi \in \mathcal{P} = \{1, \ldots, P\}$, the aim is to identify the one offering a better alternative in term of maximizing the worst possible outcome.
Herein is presented a cooperative approach based on two local search algorithms : (i) a limited-area local search applied in the elite neighborhood and which accepts the first solution with some deterioration threshold of the current solution, (ii) a wide range local search is applied to perform a sequence of paths exchange to improve the current solution.
Results have been analyzed by means state-of-the art methods and via problem instances obtained by a generator code taken from the literature. The tests were executed in compeltely comparable scenarios to those of the literature. The results are promising and the efficiency of the proposed approach is also shown.

Key words: Combinatorial optimization; Knapsack; Max-min optimization; Robust optimization; Heuristics; Cooperative
2008 MSC: 2008, 90C10, 90C35, 90C59, 90C90

1. Introduction

The Multiple-Scenario Max-Min Knapsack Problem (MSM^{2}KP) is a variant of the well known single 0-1 Knapsack Problem (KP) (e.g. Martello and Toth (1990), Martello et al. (2000)). MSM^{2}KP is a max-min binary knapsack problem with multiple scenarios. The aim of this max-min optimization problem is to maximize the minimum of a family of linear objective functions subject to a single functional constraint. Several applications of MSM^{2}KP (linear and non-linear types) match with the real world situations where the aim is to find an equilibrium for many criterias or an efficient solution. In this case the solution is said to be Pareto-optimal (see e.g. Branke et al. (2008)). However, there are several multiobjective problems where the Pareto-optimal solution building strategy is not effective enough to reach an efficient solution since the equity or fairness among uniform individual outcomes is an important issue.

This type of model is also applied when we expect to forcast a situation depending on many scenarios or simply when the objective is formulated as a maximum or a minimum of a class of criterias. As an application of this model, we can cite resources allocation (e.g. see Tang (1988)), matroid theory (e.g. see Averbach et al. (1995)), economics (e.g. see Yu (1996)), game theory (e.g. see Nash and Sofer (1996)) or multiobjective optimization (e.g. see Bitran (1977) and Ecker and Shoemaker (1981)). Recently, Taniguchi et al. (2008) have studied MSM^{2}KP and proposed an algorithm to optimally solve the problem. The method was able to solve instance problems with a number of variables up to 1000 items and 30 scenarios.

Our approach investigated this problem under the framework of the max-min optimization, where we maximize the minimum of all the objectives. In some literature it is reffered to as the robust optimization (e.g. see Yu (1996)). While, in general, such a multiobjective optimization problem is considered as a multi-criteria decision making (e.g. see Steuer (1986)). We recall that the concept of robust optimization finds its origins in engineering, specifically, control theory.

Let $\mathcal{P}$ be the set of scenarios describing all possible payoffs. We assume that each payoff might occur with some positive and unknown probability. Then maximizing the minimal payoff $Z^{\pi}(X)$, $\forall \pi \in \mathcal{P}$ and X is the binary solution vector related to the scenario π can be seen as "reasonable-worstcase optimization" and where the decision-maker protects revenue against the worst-case values. Hence, the max-min optimization approach is an alternative method to stochastic programming which can be applied for any problem whose parameter values are unknown, variable, and their distributions are uncertain. The max-min optimization consists of computing the cost of robustness of the robust counterpart to the maximum outcome optimization problem. Due to the scenario uncertainty, a max-min optimization model is likely to be more useful to a risk-averse decision maker.

In spite of its comparatively long history (e.g. Gilmore and Gomory (1966)), many variants of the knapsack problem are still being studied intensively. In this paper, we are concerned with the Multiple-Scenario Max-Min Knapsack Problem; namely MSM^{2}KP. This problem generalizes the classical knapsack problem with respect to: (i) each item $j \in J$ is associated with P values $(v_j^\pi)_{(j=1,\ldots,n)}^{(\pi=1,\ldots,P)}$, for which each one correponds to a different scenario, and a single weight w_j, $j = 1, \ldots, n$, (ii) instead of trying to compute a single solution with a maximum total profit, we compute a set of feasible solutions of different values and covering all the possible scenarios $\pi = 1, \ldots, P$. Then the worst scenario is identified in order to perform a maximization.
Given a knapsack of a fixed capacity C, the MSM^{2}KP can formally be stated as follows:

$$(MSM^2KP) \begin{cases} \max\limits_{X} & Z(X) = \min\limits_{1\leq\pi\leq P} \Big\{ \sum\limits_{j\in J} v_j^\pi x_j \Big\} \\ \text{Subject to:} & \sum\limits_{j\in J} w_j x_j \leq C \\ & x_j \in \{0,1\}, \text{ for } j = 1, \ldots, n \end{cases}$$

$X = (x_1, \ldots, x_j, \ldots, x_{n=|J|})$, the variable x_j is either equal to 0, implying item j is not picked, or equal to 1 implying item j is picked, knowing the worst scenario $\pi^\star$.

We may assume without loss of generality that w_j, v_j^π (for $j = 1, \ldots, n$ and $\pi = 1, \ldots, P$) and C are positive integers, and $\sum_{j\in J} w_j > C$.

In the following, the items $j \in J$, for each scenario $\pi = 1, \ldots, P$ are sorted regarding a specific order based on the efficiency of the items and which we detail in section 3. This order-based efficiency determines priority to some items to select in the solution. Notice that if two efficiencies (for the same scenario π) have the same value, then we consider, in first, the efficiency corresponding to the greater profit. The so-defined problem is NP-hard since for $P = 1$, it simply represents the single Knapsack Problem (KP) which is NP-hard (see Martello and Toth (1990)).

The remaining of the paper is organized as follows. First (Section 2), we briefly review some related works on the classical knapsack problem and some of its variants, also on the max-min (or min-max) combinatorial problems dealing with sequential exact and approximate algorithms. Second (Section 3), we detail the algorithm which ensures the feasiblity to the obtained solution. This procedure consists to construct, in a greedy way, a feasible sub-solution to complete by adding items realizing the minimum objective value without exceeding the remaining capacity. This algorithm identifies some critical elements j regarding the current scenario $\pi^\star$ for the solution such that $\sum_{k=1}^{j-1} w_k \leq C < \sum_{k=1}^{j} w_k$ which allow to diversify the search in the neighborhood. Then the approach will apply some complex local search strategies to attempt some improvements.

The purpose of this heuristic is to find a starting feasible solution to MSM^{2}KP in order to evaluate the influence of the initial solution's quality in the overall performance of the cooperative search. In Section 4, we detail the cooperative approach. It consists to improve the current solution by enhancing the local search around several types of feasible neighbrohood search space. The neighborhoods selection is decided regarding a certain criterion and the main local search is run regarding the type of the selected neighborhood. So that it can be seen as a hierarchical depth search around the solution neighborhood. In Section 5, we present several series of computational results on problem instances obtained by a generator code taken from the literature. We show the efficiency of the proposed approach regarding the obtained solutions. Finally, in the conclusion (Section 6), we summarize the main results of the paper and give a conclusion.

2. Literature survey

The Knapsack Problem (KP) is an NP-hard combinatorial problem (for more details on complexity theory the reader may refer to Garey and Johnson (1979)) and has been widely studied in the literature. The way of seeking a solution depends on the particular framework of the application (leading to the particular knapsack problem) and the available computational resources. Hence, a good review of the single knapsack problem and its associated exact and heuristic algorithms is available in Martello and Toth (1990). For the (un)bounded single constraint KP, we can find in the literature a large variety of solution methods (see Martello et al. (1999); Balas and Zemel (1980); Fayard and Plateau (1982) and Pisinger (1997)). The problem has been solved optimally and approximately by dynamic programming, search tree procedures or other hybrid approaches.

Several approaches have also been developed in other previous works that adressed the general case, i.e., when the number of constraints is extended to more than one or the aim is to optimize a class of functions (multiobjective KP). A first variant of the KP is called Multidimensional (or Multiconstraint)

Knapsack Problem (MDKP) (see Chu and Beasley (1998)). The Multiple-Choice Knapsack Problem (MCKP) (see Pisinger (1995)) is another variant where the picking criterion of items is more restrictive. Another more harder variant is the Multiple-choice Multidimensional Knapsack Problem (MMKP) (see Hifi et al. (2006) and Sbihi (2007)). A recent detailed review for the MMKP is given in Sbihi (2007). If we deal with only two knapsack constraints, then the problem is referred to as the Bidimensional Knapsack Problem (BKP) (see Freville and Plateau (1997)). Another type of combinatorial optimization problem is the max-min (min-max) allocation problem. This problem has widely been studied in the literature (see Brown (1991); Luss (1992); Pang and Yu (1989) and Tang (1988)). We also can find in the literature another max-min optimization problem of knapsack type which has been aproximately and optimally solved via tabu search based-heurstic, branch and bound and binary search methods (see Hifi et al. (2002) and Yamada et al. (1998)).

To the best of our knowledge, there exists only three papers dealing directly or indirectly with the Multiple-Scenario Max-Min Knapsack Problem (MSM^{2}KP) (see Yu (1996); Iida (1999) and Taniguchi et al. (2008)). The first paper (Yu (1996)) proposed to optimally solve MSM^{2}KP by a branch and bound algorithm. The approach was able to solve instance problems up to 60 items. In the second paper (Iida (1999)), the author developped a method to obtain new lower and upper bounds for the max-min 0-1 knapsack problem thanks to a derivative procedure-based lagrangean relaxation. The approach used also a branch and bound procedure developped in Yu (1996) as well as the obtained lower and upper bounds. The author showed that these bounds were good enough to solve the problem in a acceptable computing time. However, the method failed to increase the total number of items more than 60. Finally, in Taniguchi et al. (2008), the paper developped an approach in two steps to optimally solve MSM^{2}KP. First, the authors proposed a surrogate relaxation heuristic to reduce the problem, then they compute some upper and lower bounds. This technique allowed to reduce the problem and permits to optimally solve the remaining problem by a branch and bound procedure. In this paper, we analyze and present an alternative optimization method. It can be seen as a first step to extend to more than a simple method which iteratively attempts to select a good heuristic amongst many.

3. A greedy algorithm for the MSM^{2}KP

Prior to design our main approach, we propose an algorithm called herein GH in order to build a starting feasible solution. This initial solution is obtained iteratively and the aim is to perform better improvement throughout the process by a very sophisticated method.

Let $X = (x_j)_{j=1,\ldots,n}$ be a solution vector, where $x_j = 1$ if item j is selected in the solution, and $x_j = 0$ otherwise. The aim is to maximize the total profit regarding the minimal π-objective value $Z^\pi(X)$ obtained over all the P scenarios.

In this heuristic, items are initially sorted according to a specific decreasing order. For each item $j = 1, \ldots, n$, we compute first its efficiency regarding the scenario π as $e_j^\pi = v_j^\pi / w_j$. We say that $(j \prec k)$ iff $e_j^\pi \geq e_k^\pi$. Considering this order, we apply a reordering of the total items as a preprocessing step.

Then, GH builds iteratively a partial feasible solution. Indeed, in the main steps, for each selected item j, the procedure attempts to complete the solution. The procedure locates a minimal scenario $\pi^\star$ and computes $Z(\overline{X}) = \max_x \sum_{j\in J} v_j^{\pi^\star} x_j$. GH terminates once a feasible solution $Z(\overline{X})$ is obtained. We recall that the obtained solution could be of bad quality. Herein, we describe the main steps of the greedy algorithm GH :

Algorithm GH (Greedy Heuristic)
Input: An MSM^{2}KP instance I
Ouput: A feasible solution $\overline{X}$ for MSM^{2}KP

Initialization:

1. `FOR` $j = 1, \ldots, n$ and $\pi = 1, \ldots, P$ `DO`
   `Sort` items in the decreasing order of the efficiency e_j^π
2. $\pi^\star = 1$; X^0: initial; //$\pi^\star$: scenario with the minimum (sub)solution
3. `Set` $\overline{X} \leftarrow X^0$; $Z(\overline{X}) \leftarrow Z^{\pi^\star}(X^0)$; //$\overline{X}$: feasible starting point e.g. 0

Main steps:

4. `REPEAT`
5. `IF` $w_j \leq (C - \sum_{k\leq j-1} w_k)$ `THEN` //the solution is feasible for $\pi^\star$
   $Z^{\pi^\star}(\overline{X}) \leftarrow Z^{\pi^\star}(\overline{X}) + v_j^{\pi^\star}$;
   $j \leftarrow j + 1$;
   `Let` $\pi^\star = \arg\min_{1\leq\pi\leq P} \{Z^\pi(\overline{X})\}$;
6. `UNTIL` $j > |J|$
7. `EXIT` with best feasible solution $\overline{X}$ of best value $Z(\overline{X})$;

4. A cooperative local search

In this section, we present the main principle of the cooperative local search-based approach namely CLS. The main algorithm contains several steps. Tabu search starts by a feasible solution obtained thanks to GH. All the visited solutions are feasible. The exploration of the solutions space is executed with some add/drop swaps. The elite solutions list is generated by improving the objective value where the minimal scenario has already been identified. The core of the approach is to build neighborhoods and perform several local searches in order to reach a best near-optimal solution.

Our cooperative strategy takes place when a search procedure gives away information about its best solutions and another procedure adapts its search trajectory based on this information. The cooperative local search framework is composed of a decision parameter and of two local search heuristics (notice that we can extend the method to more than 2 local search heuristics). The decision parameter is in charge of the selection and the execution of the appropriate local search heuristic at each decision point of the search process. The greedy strategy selects the best, not necessarily an improving, scenario. The tabu based strategy, incorporates a tabu list in the selection mechanism that forbids the selection of the non-improving solution for a certain tabu tenure. The decision parameter allows to accept the selected local search heuristic which accepts only to improve the current solution or either to improve the current solution or to lead to the smallest deterioration of the current solution if no improvement can be achieved.

The aim of using the cooperative heuristic is to raise the level of generality so as to be able to apply the same solution method to several criteria problems. Perhaps at the expense of reduced but still acceptable solution quality when compared to a tailor-made approach. Hence, we define two types of local search that address MSM^{2}KP neighborhood search : (i) a generalized search outlined herein by GS and (ii) a limited search represented herein by RS.

The GS algorithm is a very large local search and applied if the list L has not included enough moves regarding a certain threshold S. Otherwise the RS algorithm is called in order to perform a limited-area local search in the elite neighborhood. RS accepts the first solution which slightly deteriorates the current solution $X^{current}$. The limited-area local search is supposed to perform in a very rich elite neighborhood.

To ensure the feasibility of the solution, we apply a feasible state phase regarding the best recorded infeasible solution. It corresponds to set to '`zero`' the solution components x_j $(j = 1, \ldots, n)$ which are equal to '`one`' and have the lowest efficiency $e_j^{\pi^\star}$.

Algorithm FSH (Feasible State Heuristic)

Initialization:

1. Set $\overline{X} \leftarrow \underline{X}$ $//\underline{X}$ is the recorded best infeasible solution

Main steps:

2. REPEAT
3. Select $j_{min} = \underset{1 \le j \le n}{\text{argmin}} \{e_j^{\pi^\star} \mid \overline{x}_j = 1\}$
4. $\overline{x}_{j_{min}} \leftarrow 0;$
5. UNTIL $\sum_{1 \le j \le n} w_j \overline{x}_j \le C;$
6. END

The tabu approach uses a tabu list L that includes all the tabu moves. The tabu status for these moves is amended regarding a period τ such that $\alpha\sqrt{n} \le \tau \le 2\alpha\sqrt{n}$, ($\alpha \in [0.5; 1.5]$ and n is the problem size). The tabu status of a move is removed if it belongs to the list L and it exceeds τ iterations, or if it matches with the aspiration criteria.

4.1. Intensification and diversification

It is achieved via GH which is called once an improvement of the current solution is realized. We set L an elite moves list and $\pi^\star = \underset{1 \le \pi \le P}{\text{argmin}} \{\sum_{j \in J} v_j^\pi x_j\}$ is the index corresponding to the minimal scenario corresponding to the current solution $X^{current}$.

4.1.1. The very large local search

The very large local search or generalized search (algorithm GS) starts with a feasible solution and performs a sequence of path exchange to improve the solution. The list L contains the moves that improves the objective value $Z(.) = Z^{\pi^\star}(X)$ and $j \in L \Leftrightarrow (v_j^{\pi^\star} \ge 0$ and $x_j = 1)$.

Algorithm GS (Generalized Search)

Initialization:

1. L=$\{j_1, j_2, \ldots, j_r\}$; $j^\star = j_1$;
2. FOR $\pi = 1$ to P DO
3. $\quad Value^\pi = \sum_{j \neq j^\star} v_j^\pi x_j + v_{j^\star}^\pi (x_{j^\star} \pm \epsilon_{j^\star})$; $\epsilon_j = \begin{cases} +1 & if \quad \overline{x}_j = 0 \\ -1 & if \quad \overline{x}_j = 1 \end{cases}$;
4. $\mu = \min_{1 \leq \pi \leq P} \{Value^\pi\}$;

<u>**Main steps:**</u>

5. FOR $j = j_2$ to j_r DO
6. $\quad \pi \leftarrow 1$; $m = \sum_{k \neq j} v_k^\pi x_k + v_j^\pi (x_j \pm \epsilon_j)$;
7. WHILE ($Value^\pi \geq \mu$ and $\pi \leq P$) DO //abandon candidate j once it leads to a smaller value of μ
8. $\quad \pi \leftarrow \pi + 1$;
9. $\quad$ IF ($Value^\pi \leq m$) THEN
10. $\qquad m \leftarrow Value^\pi$; //save the minimum of all $Value^\pi, \pi = 1, \ldots, P$
11. END_WHILE
12. IF $\pi = P$ THEN $j^\star \leftarrow j$; $\mu \leftarrow m$;
13. END

Furthermore, we consider a set L_{swap} of couples of items (j_1, j_2) that are candidates for a swap such that the exchange operation allows to improve the current value of the solution.

$$(j_1, j_2) \in L_{\text{swap}} \Leftrightarrow \left(\begin{cases} v_{j_1}^{\pi^\star} \geq 0;\ x_{j_1} = 1 \\ \text{and} \\ x_{j_2} = 0 \end{cases} \text{or} \begin{cases} v_{j_2}^{\pi^\star} \geq 0;\ x_{j_2} = 1 \\ \text{and} \\ x_{j_1} = 0 \end{cases} \right)$$

This procedure performs systematically a search of all the solutions contained in the list L.

4.1.2. The restricted guided local search

Contrary to GS algorithm, RS algorithm is a targeting limited-area local search. It allows to make savings in term of locating the best solution or to accept degrading solution with some threshold. This is achieved via a filtering process thanks to a decreasing threshold acceptance function $\Phi(\theta)$, where θ is a parameter to set up. The threshold acceptance function $\Phi(.)$ can be seen as a penalization-based concept. We detail in what follows the RS algorithm :

Algorithm RS (Restricted Search)

Initialization:

1. $L=\{j_1, j_2, \ldots, j_\ell\}$; $j^\star = j_1$, and $\ell > S$;
   $//L$ is a solution subspace with ℓ 1-components
   $//S$ is a parameter controlling the search complexity
2. $Value^\pi = \sum_{j \neq j^\star} v_j^\pi x_j^{current} + v_{j^\star}^\pi (x_{j^\star}^{current} \pm \epsilon_{j^\star})$, $\pi = 1, \ldots, P$;
3. $\mu = \min\limits_{1 \leq \pi \leq P} \{Value^\pi\}$;

Main steps:

5. `WHILE` (not Stopping Condition `and` $k \leq S$) `DO`
6. $\quad j_k \leftarrow$ `random`(j, L); $L \leftarrow L \setminus \{j\}$; //random selection of item j
7. $\quad Value^\pi = \sum_{j \neq j_k} v_j^\pi x_j^{current} + v_{j_k}^\pi (x_{j_k}^{current} \pm \epsilon_{j_k})$, $\pi = 1, \ldots, P$;
8. $\quad \mu = \min\limits_{1 \leq \pi \leq P} Value^\pi$;
9. $\quad$ `IF` $\mu \geq Z(X^{current}) + \Phi(\theta) \times Z(X^{current})$ `THEN` $//Z(.)$ best value
10. $\qquad j^\star \leftarrow j_k$;
11. $\quad$ `ELSE` $k \leftarrow k + 1$;
12. `END_WHILE`
13. `END`

The algorithm starts by setting the current solution to a selected one belonging to the region L with $|L| = \ell$. Then it applies a restricted swaps sequence in order to remain in the same region to better explore it. Another feature is to guide the search to unexplored or not enough explored regions thanks to a changing region movement and some swaps to explore the new region.

The idea is to accept less and less degraded solutions progressively of the search process. It uses memory to guide the search to promising regions of the solution space. This is performed by increasing the current cost function with a penalty term that penalizes bad features of previously visited solutions. However, $\Phi(.)$ can trap the local search around some local optima.

This reduction in the search spaces enables a fast execution of the global algorithm. Unfortunately, it also imposes serious limitations on the ability to provide good quality intermediate solutions.

4.1.3. An improved 2-phase tabu search for MSM^{2}KP

The concept of the approach is to help guide the search towards the optimization of the different individual objectives. It starts with a feasible solution obtained thanks to the GH algorithm. For the local search, we apply both large and limited local search (algorithm GS and algorithm RS). It starts with an initial complete solution and then makes a request for informations to perform the right local search using the assigned heuristic either for a certain number of iterations, or until no further improvement is possible. The process performs search through the same solution space, starting from the same current solution. If the obtained solution is not feasible, then the feasible state phase (algorithm FSH) is applied to convert the solution in order to build a new solution regarding the best solution found in the list L. This operation permits also to release the solution trapped around a local optima.

Algorithm CLS (Cooperative Local Search)
Input: An MSM^{2}KP instance I
Ouput: A best near-optimal solution $X^\star$ for MSM^{2}KP

Initialization:

1. $\overline{X} = (\overline{x}_1, \ldots, \overline{x}_j, \ldots, \overline{x}_n)$:= `GH()`;//a feasible initial solution
2. $X^{current} \leftarrow X^\star \leftarrow \overline{X}$;
3. $Iter \leftarrow 0$; $L \leftarrow \emptyset$;
4. $Z(X^{current}) \leftarrow Value^{current} \leftarrow \min_{1\leq\pi\leq P}\{\sum_{j\in J} v_j^\pi x_j^{current}\}$;

Main steps:

5. `FSH()`$\longleftarrow$`FALSE`; //the solution is feasible
6. $\pi^\star=\arg\min_{1\leq\pi\leq P} \sum_{j\in J} v_j^\pi x_j^{current}$; //if many, select the first obtained one
7. $L \longleftarrow$ `construct()`; //generate the list of moves elite
8. $t \leftarrow 0$; //internal counter for the local search
9. `IF` $|L| \leq S$ `THEN`
   GS($x^{current}, j^\star$);
   $x_{j^\star}^{current} \leftarrow x_{j^\star}^{current} \pm \epsilon_{j^\star}$;
   $t \leftarrow t+1$;
10. `IF` $|L| > S$ `THEN`
   RS($x^{current}, j^\star$);
   $x_{j^\star}^{current} \leftarrow x_{j^\star}^{current} \pm \epsilon_{j^\star}$;
   $t \leftarrow t+1$;
11. $Value^{current} \leftarrow \min_{\pi}\{\sum_{j\neq j^\star}(v_j^\pi x_j^{current}) + v_{j^\star}^\pi x_{j^\star}^{current}\}$;
12. `IF` $(Value^{current} \geq Z(X^{current})$ `and FSH()`$)$ `THEN`
   $Z(X^{current}) \leftarrow Value^{current}$;
   $X^\star \leftarrow X^{current}$;
13. `UPDATE` L, $Iter$;
14. `IF` $Iter \geq Max\_Iter$ `THEN` STOP;
15. `EXIT` with solution $X^\star$ of value $Z(X^\star)$;

We detail the method as a chosen heuristic to guide the search towards the desired regions of the trade-off area. It takes into account the localization of the current solution in the objective space and the ability of each neighbourhood exploration heuristic to achieve improvements on each of the individual objectives. The local search process allows to run the selected heuristic to the current solution for a certain number of iterations, or until no further improvement is possible.

The startegy attempts to *restrict* the likelihood of generating solutions in *crowded* regions of the trade-off surface and *enhance* the likelihood of generating solutions in *under-populated* regions. The algorithm using this strategy, attempts to perform a more "intelligent" path finding by applying the neighbourhood search heuristic that is more likely to "guide" the solution in the desired direction.

5. Computational results

We have run initial tests to analyze the computational performances of both greedy heuristic GH and the developed approach CLS. The proposed algorithm is coded in C++ and run on `Sony VAIO Centrino Duo Core` laptop (with 1.66 Ghz and 1 GB/Go of Ram). Our computational study was conducted on 132 problem instances of various sizes and densities. These test problem instances (detailed in Table 0) are standard and their optimal solution values are not known.

First of all, we notice that we did not succeed to have access to the same problem instances tested by Taniguchi et al. (2008). We only had access to the problems generator used by these authors who provided us with the code. We may also notice that since we used a different computer with a different processor than the one used by Taniguchi et al. (2008), the generator code generated problem instances with different values but the design technique remains the same.

The problem instances are generated as follows: each weight w_j and the ordinal profit v_j^0 $(j = 1, \dots, n)$ are randomly, uniformly and independently generated in the integer interval $[1, 100]$. For a given scenario π, the value v_j^π of each item j is uniform and random integer in the interval $[(1-\sigma)v_j^0,\ (1+\sigma)v_j^0]$, where $\sigma \in \{0.3;\ 0.6;\ 0.9\}$ is a parameter to set up the correlation level between different scenarios. For instance, more σ is close to 0, more the scenarios are strongly correlated and consequently the profits are also strongly correlated. The knapsack capacity is set to $C = \frac{1}{D}\sum_{1\leq j\leq n} w_j$. D is constant to set.

These instances are divided into three different groups depending on the σ value. The first group ($\sigma = 0.9$) represents the "uncorrelated" or "likelyhood uncorrelated" instances, the second one ($\sigma = 0.6$) contains the "weakly correlated" instances and the last group ($\sigma = 0.3$) represents the "strongly correlated" instances. In addition to this, we set D the availability parameter that indicates whether the capacity C is tight enough or not. Practically, we designed our computational test protocol by setting three types of sets S_1, S_2 and S_3.

Class	Inst.	n	P	Inst.	n	P
S_1	`I01x.D`	1000	40	`I02x.D`	1500	40
	`I03x.D`	2000	40	`I04x.D`	2500	40
	`I05x.D`	3000	40	`I06x.D`	3500	20
	`I07x.D`	4000	20	`I08x.D`	4500	20
	`I09x.D`	5000	20	`I10x.D`	6000	20
	`I11x.D`	10000	20			
S_2	`I12x.D`	2500	50	`I13x.D`	3000	50
	`I14x.D`	3500	50	`I15x.D`	5000	30
	`I16x.D`	8000	20			
S_3	`I17x.D`	500	60	`I18x.D`	1000	60
	`I19x.D`	1500	60	`I20x.D`	2000	60
	`I21x.D`	200	80	`I22x.D`	200	100

Table 0: Test problem instances details: x=u, w, s; D=2, 4

Here `u` ($\sigma = 0.9$) denotes uncorrelated problem instances, `w` ($\sigma = 0.6$) denotes weakly correlated problem instances and `s` ($\sigma = 0.3$) denotes strongly correlated problem instances.
On one hand, S_1 includes problem instances with relatively a big number of items n and a small number of scenarios P, S_2 contains problem instances with a relative medium n and P and on the other hand, S_3 contains problem instances with a relative small n and a big P.

5.1. The results summary

The purpose of this section is twofold: (i) to evaluate the performance of the GH and CLS and (ii) to determine the best trade-off between the running time and the used parameters for the cooperative CLS algorithm: the maximum number of iterations, the length of the list L and the penalty $\Phi(.)$ function. Each parameter of CLS algorithm has been calibrated in order to obtain the best possible performances for each problem. The length of the tabu list L varies dynamically. Indeed, if P is the number of the different scenarios, then the parameter S is automatically and randomly taken in the interval $[\lfloor\sqrt{P}\rfloor + 25, \lfloor\sqrt{P}\rfloor + 35]$. Other parameters settings were necessary. We summarize them in the table 1.

Iter	α	τ	θ	$\Phi(\theta)$
4500 (*uncor*) 2500 (*w_cor*) 1500 (*s_cor*)	0.9 (*uncor*) 1.1 (*w_cor*) 1.3 (*s_cor*)	random($[\alpha\sqrt{n}, 2\alpha\sqrt{n}]$)	random[1,10]	$\frac{\theta}{(Iter)^{3/2}}$

Table 1: Parameters settings

In a preliminary experiment, we have solved the problem instances by considering several parameters of the algorithm. The details of the test results appear in Tables 2-10.

To analyze the method's performance, we first compute the upper bound UB (we used Dantzig (1957)'s upper bound) to determine : (i) the gap $\gamma := (UB - Z(X^\star))$ between UB and the best obtained solution value $Z(X^\star)$ and (ii) the average percentage deviation $\rho := 100.(\frac{UB - Z(X^\star)}{Z(X^\star)})$, since we don't know the optimal solution for the treated problem instances. We explain in the Upper bound computation Appendix how to compute UB.

In our numerical tests, we solved the same problem instances sets for each fixed parameter D, α, τ, $Iter$ and $\Phi(.)$.

We may notice that we have improved these results by increasing the size of the problems regarding the number of the items as well as the number of the scenarios. CLS is able to process a very large number of variables within a few amount of cpu consuming time. In some cases, the algorithm is either able to lead to the optimal solution. We remark that for several problems of different sizes and types and belonging to S_1 and S_2, the obtained solution $Z(X^\star)$ is equal to the upper bound UB. In this case the algorithm has reached the optimal solution since it is the biggest feasible solution. The percentage of these obtained solutions depends on the set we have considered.

Furthermore, the algorithm is able to produce high quality near-optimal solutions for these sets ($n = 10000$ and $P = 100$ scenarios) within a relatively small cpu consuming time. While in Taniguchi et al. (2008), the approach was able to solve the problem with a number of items n less than 1000 and a number of scenarios P less than 30 and in less than a few seconds. Indeed, these solutions were obtained within 1% of relative errors. Our method has solved approximately larger problems and faster regarding the size of the treated problems. Comparing these obtained results to those of Taniguchi et al. (2008) and based on the same problems design, we can remark that our approach is able to lead to solutions of good quality for the big size problems (in term of number of items and scenarios).

On average, the first set S_1 (for uncorrelated, weakly and strongly correlated problems) gives the highest amount of reached optimal solutions by CLS and particularly for the uncorrelated set with 13 solutions out of 22 problems. The average running time is less than one minute and does not exceed 22 *sec*. The average relative percentage error is less than 1% and is located in the intervals $[0, 0.08\%]$ for the uncorrelated, $[0, 0.221\%]$ the weakly correlated and $[0, 0.073\%]$ the strongly correlated problems. Furthermore, the algorithm is able to produce high quality near-optimal solutions for these sets ($n = 10000$ and $P = 40$ scenarios) within a relatively small cpu consuming time.

Inst.	GH	UB	$Z(X^\star)$	γ	$\rho\%$	$CPU(s)$
I01u.2	40725	40740.00	40740	0	0	2.54
I01u.4	31328	31427.17	31403	24.17	0.076	27.2
I02u.2	61173	61218.00	61218	0	0	2.15
I02u.4	44147	44226.93	44207	19.93	0.045	2.78
I03u.2	82818	82830.00	82830	0	0	2.53
I03u.4	58872	61156.00	61156	0	0	3.31
I04u.2	104025	104044.34	104043	1.34	0.001	3.51
I04u.4	73877	73951.44	73911	40.44	0.054	3.78
I05u.2	123280	123297.72	123297	0.72	0	2.95
I05u.4	90817	90946.45	90939	7.45	0.080	3.72
I06u.2	146094	146118.00	146118	0	0	3.28
I06u.4	103424	103472.00	103451	21	0.020	3.81
I07u.2	163523	163529.00	163529	0	0	3.75
I07u.4	119007	119114.32	119088	26.32	0.022	3.86
I08u.2	187034	187040.00	187040	0	0	3.62
I08u.4	135952	136004.00	136004	0	0	3.77
I09u.2	208114	208136.00	208136	0	0	4.12
I09u.4	148808	148830.00	148830	0	0	4.19
I10u.2	256397	256416.23	256415	1.23	0	4.91
I10u.4	191406	191491.00	191491	0	0	5.15
I11u.2	428807	428849.00	428849	0	0	5.68
I11u.4	318754	318813.00	318813	0	0	7.73

Table 2: Uncorrelated set S_1 results

The second set S_2 of computational experiments (uncorrelated, weakly and strongly correlated) gives an overview of the algorithm behaviour regarding both the obtainded upper bound and solution. The algorithm reached 8 optimal solutions out of 10 problems. For this set, the results present the same quality regarding the running cpu time which is less than 24 *sec* and the worst percentage deviation $\rho = 0.082\%$. The average relative percentage error is located in the intervals $[0, 0.033\%]$ for the uncorrelated, $[0, 0.0083\%]$ the weakly correlated

Inst.	GH	UB	$Z(X^\star)$	γ	$\rho\%$	$CPU(s)$
I01w.2	41095	41172.64	41133	39.64	0.096	3.21
I01w.4	28343	28446.76	28384	62.76	0.221	3.39
I02w.2	60256	60328.80	60291	37.8	0.062	3.54
I02w.4	44609	44654.00	44654	0	0	3.67
I03w.2	79872	79936.00	79896	40	0.050	3.81
I03w.4	58854	58864.16	58863	1.16	0.001	4.25
I04w.2	100545	100677.94	100566	111.94	0.111	4.43
I04w.4	71988	72108.92	72093	15.92	0.022	4.72
I05w.2	120991	121031.44	121030	1.44	0.001	5.37
I05w.4	86756	86837.35	86821	16.35	0.018	5.56
I06w.2	140795	14837.00	14837	0	0	5.84
I06w.4	99827	99861.00	99861	0	0	6.17
I07w.2	162739	162756.56	162755	1.56	0	6.28
I07w.4	113325	113367.00	113367	0	0	6.63
I08w.2	180524	180538.70	180538	0.7	0	6.72
I08w.4	130718	130801.93	130744	57.93	0.044	7.14
I09w.2	202081	202139.00	202099	40	0.019	7.23
I09w.4	143549	143806.18	143699	107.18	0.074	7.31
I10w.2	245519	245549.56	245547	2.56	0.001	7.42
I10w.4	178630	178658.00	178655	3.32	0.001	7.66
I11w.2	416328	416378.12	416371	7.12	0.001	8.23
I11w.4	303786	303944.56	303938	6.65	0.002	8.45

Table 3: Weakly correlated set S_1 results

and $[0, 0.082\%]$ the strongly correlated problems. Furthermore, this set seems to be more easy than the first set S_1 regarding the global results. Problems with medium size seems to be relatively easy to tackle.

Inst.	GH	UB	$Z(X^\star)$	γ	$\rho\%$	$CPU(s)$
I01s.2	41651	41681.89	41679	2.89	0.006	7.91
I01s.4	46909	46947.00	46947	0	0	8.24
I02s.2	63029	63083.28	63037	46.28	0.073	9.36
I02s.4	45331	45425.60	45396	29.6	0.065	9.76
I03s.2	82378	82426.34	82417	9.344	0.011	9.97
I03s.4	63661	63666.65	63664	2.65	0.004	10.32
I04s.2	102602	105720.75	105715	5.75	0.005	10.73
I04s.4	76435	76502.83	76481	21.83	0.028	11.36
I05s.2	126306	126368.68	126349	19.68	0.015	13.55
I05s.4	93579	93594.97	93589	5.97	0.006	13.99
I06s.2	151932	151952.00	151952	0	0	14.36
I06s.4	110497	110511.00	110511	0	0	14.82
I07s.2	168199	168211.00	168211	0	0	15.94
I07s.4	128169	128273.46	128223	50.46	0.039	16.21
I08s.2	191228	191242.20	191235	7.2	0.003	17.34
I08s.4	143825	143917.26	143885	32.26	0.022	17.78
I09s.2	191228	211728.00	211728	0	0	18.33
I09s.4	158404	158530.73	158477	53.73	0.033	18.54
I10s.2	247302	247873.84	247855	18.84	0.007	19.35
I10s.4	172299	172381.09	172379	2.09	0.001	19.66
I11s.2	405655	405709.71	405709	0.79	0	21.45
I11s.4	290357	290417.00	290417	0	0	21.93

Table 4: Strongly correlated set S_1 results

Inst.	GH	UB	$Z(X^\star)$	γ	$\rho\%$	$CPU(s)$
I12u.2	102608	102608.00	102608	0	0	8.93
I12u.4	72767	72784.49	72782	2.49	0.003	9.12
I13u.2	122302	122315.16	122310	3.16	0.002	9.73
I13u.4	87527	88549.00	88549	0	0	11.17
I14u.2	143316	143328.00	143328	0	0	13.96
I14u.4	101067	101103.70	101070	33.7	0.033	14.23
I15u.2	217652	217652.00	217652	0	0	18.73
I15u.4	158841	158921.00	158876	45	0.028	19.34
I16u.2	343368	343399.00	343399	0	0	7.23
I16u.4	253303	253316.00	253316	0	0	8.56

Table 5: Uncorrelated set S_2 results

Inst.	GH	UB	$Z(X^\star)$	γ	$\rho\%$	$CPU(s)$
I12w.2	106592	106624.23	106623	1.23	0.001	9.35
I12w.4	79249	79286.63	79284	2.63	0.003	9.63
I13w.2	127545	127566.00	127566	0	0	10.72
I13w.4	95303	95401.82	95322	79.82	0.083	11.13
I14w.2	147832	147848.00	147848	0	0	12.58
I14w.4	108703	108786.82	108757	29.82	0.027	12.84
I15w.2	208762	208817.54	208775	42.54	0.020	13.57
I15w.4	151046	151060.00	151060	0	0	14.36
I16w.2	329312	329336.00	329336	0	0	9.92
I16w.4	242503	242574.45	242570	4.45	0.001	11.06

Table 6: Weakly correlated set S_2 results

Inst.	GH	UB	$Z(X^\star)$	γ	$\rho\%$	$CPU(s)$
I12s.2	103173	103180.00	103180	0	0	11.24
I12s.4	75012	75099.24	75037	62.24	0.082	11.78
I13s.2	123190	123243.25	123218	25.25	0.020	12.43
I13s.4	88446	88545.20	88502	43.2	0.004	12.84
I14s.2	142592	142661.67	142631	30.67	0.002	13.41
I14s.4	103385	103386.00	103386	0	0	13.75
I15s.2	204113	204669.42	204662	7.42	0.003	15.62
I15s.4	145587	145717.32	145709	8.32	0.005	16.85
I16s.2	325303	325375.00	325375	0	0	20.14
I16s.4	231209	231300.81	231295	5.81	0.002	23.62

Table 7: Strongly correlated set S_2 results

The third set S_3 (uncorrelated, weakly and strongly correlated problems) shows that the obtained solutions are at most 5.306% far from the upper bound. No optimal solution was reached by the algorithm. Also all the obtained solutions are computed in a big cpu consuming time comparing to the first two sets. In addition, the reached solutions are obtained with a bigger gap than those obtained in the second set S_2. The average relative percentage error is located in the intervals $[0.648\%, 4.490\%]$ for the uncorrelated, $[0.817\%, 4.039\%]$ the weakly correlated and $[0.062, 5.306\%]$ the strongly correlated problems. We remark that this set is a significantly hard set of instances (uncorrelated, weakly and strongly correlated) and the obtained solutions present less good quality than those of S_1 and S_2.

Inst.	GH	UB	$Z(X^\star)$	γ	$\rho\%$	$CPU(s)$
I17u.2	20317	21574.00	21347	227	1.063	3.92
I17u.4	15651	16538.00	16236	302	1.860	4.15
I18u.2	40837	42144.42	41873	271.42	0.648	4.53
I18u.4	30668	33476.80	32789	687.8	2.097	4.72
I19u.2	62433	66216.33	65129	1087.33	1.669	5.56
I19u.4	47511	49674.00	48973	701	1.431	5.13
I20u.2	85301	89940.66	88126	1814.66	2.059	5.42
I20u.4	60937	65922.50	65028	894.5	1.375	5.59
I21u.2	7904	8282.54	8125	157.54	1.938	6.46
I21u.4	5749	6078.92	5893	185.92	3.154	6.87
I22u.2	5736	5982.63	5843	139.63	2.389	7.15
I22u.4	3740	4103.34	3927	176.34	4.490	9.35

Table 8: Uncorrelated set S_3 results

Consequently, it is relatively easy to solve problem instances with a relatively big n and a small P or medium size of both n and P. While n is of small size and P of big size, the algorithm reaches less good quality solutions and what ever if they are uncorrelated, weakly or strongly correlated problems. In average, the algorithm is able to compute these solutions in an acceptable cpu consuming time. S_2 contains problem instances with n up to 8000 and $P = 50$ and contrary to the set S_3, the obtained solutions are of better quality regarding the computed percentage deviation error ρ. We recall that S_3 contains problem instances with n up to 2000 and $P = 100$.

The interaction between scenarios becomes more complex and the algorithm needs more processing to compute the solution. The correlation between items is also a parameter which impacts the quality of the solution.

Also, it appears that if the total of iterations is a small one, it is better to not allow the intensification and diversification. We can explain this by the fact that the recorded informations are not representative enough of the solutions space.

Inst.	GH	UB	$Z(X^\star)$	γ	$\rho\%$	$CPU(s)$
I17w.2	20589	21721.72	21223	498.72	2.349	5.61
I17w.4	14924	16534.00	15892	642	4.039	5.96
I18w.2	40675	43573.22	42527	1046.22	2.460	6.25
I18w.4	29051	32251.93	31175	1076.93	3.454	6.74
I19w.2	61013	64841.00	63794	1047	1.641	7.22
I19w.4	44024	47085.70	6154	931.7	2.020	7.53
I20w.2	82352	86474.00	85105	1369	1.610	7.96
I20w.4	58812	63230.50	61772	1458.5	2.361	8.84
I21w.2	7623	7796.23	7733	63.23	0.817	9.16
I21w.4	4645	4835.35	4735	100.35	2.119	10.27
I22w.2	5552	5776.49	5663	113.49	2.004	11.65
I22w.4	4020	4223.52	4140	83.52	2.017	12.56

Table 9: Weakly correlated set S_3 results

Inst.	GH	UB	$Z(X^\star)$	γ	$\rho\%$	$CPU(s)$
I17s.2	20207	21544.00	21256	488	2.317	7.91
I17s.4	13966	14848.73	14217	631.73	4.443	8.24
I18s.2	40932	43941.00	42345	1596	3.769	8.63
I18s.4	28979	31025.59	30453	572.59	1.880	8.97
I19s.2	61799	63727.22	62236	1491.22	2.396	9.23
I19s.4	43197	46090.00	44987	1103	2.451	9.56
I20s.2	79741	85187.50	84254	933.5	1.107	9.92
I20s.4	58115	63514.00	62478	1036	1.658	10.46
I21s.2	7702	8150.74	8012	138.74	1.731	10.93
I21s.4	5189	5381.37	5378	3.37	0.062	11.56
I22s.2	5123	6152.00	5842	310	5.306	12.53
I22s.4	2912	3630.42	3625	5	0.137	13.58

Table 10: Strongly correlated set S_3 results

6. Conclusion

In this article, we have investigated the max-min binary knapsack with multiple scenarios (MSM^{2}KP). Finding optimal solutions for the Multiple-Scenario Max-Min Knapsack Problem (MSM^{2}KP) becomes a challenging and important issue due to its complex structure and possibly large size problems to tackle. An approximate algorithm might be an appropriate way to seek near-optimal solutions in an acceptable consuming cpu. To do so, we have developed a cooperative approximate algorithm. The approach is mainly based upon tabu search and a combination of two cooperative procedures; a generalized and a restricted local search. The principle of the method is to identify the scenario $\pi^\star$ realizing the minimum total profit as a current solution and to tailor a neighborhood search to improve the obtained solution. We have used a spreading search strategy and designed a heuristic feasibility in order to improve the performance of the algorithm. Computational results showed that the two cooperative procedures applied together are able to generate high-quality solutions for the Multiple-Scenario Max-Min Knapsack Problem, within a reasonable computing time.

Our approach investigated MSM^{2}KP under the framework of the max-min optimization, where we maximize the minimum of all the objectives. As a result, we were able to approximately solve the problem with n up to 10000 variables and P up to 100 scenarios and in less than one minute. The obtained solutions were usually within 0.221% for S_1, 0.083% for S_2 and 5.306% for S_3 of relative percentage deviation errors.

In the local search, each neighborhood solution is evaluated at worst in $\mathcal{O}(nP^2)$ time and the neighborhood search is pruned heuristically by the neighbor list. During the search, infeasible solutions are allowed to be visited while the amount of violation is penalized. The computational results on a representative benchmark instances indicate that the proposed algorithm is efficient enough to tackle problem instances of a certain size (in term of number of items and scenarios). The algorithm was able to reach 30 solutions which are equal to the computed upper bound UB among a total of 132 problem instances.

Acknowledgments

The author thanks the three anonymous referees for their helpful comments and insightful suggestions on previous versions of this paper. Also, the author would like to thank Pr. Yamada for his help in providing him with his generator code that was necessary to successfully achieve the computational tests.

References

Averbakh I, Berman O, Punnen A.P (1995) Constrained matroidal bottleneck problems, Disrete Applied Mathemtics 63:201–214

Balas E, Zemel E (1980) An algorithm for large zero-one knapsack problem, Operations Research 28:1130–1154

Bitran G.R (1977) Linear multi-objective programs with zero-one variables, Mathematical Programming 13:121–139

Branke J, Deb K, Miettinen K, Slowinski R (eds.) (2008) Multiobjective Optimization: Interactive and Evolutionary Approaches. State-of-the-Art, Book Series of the Lecture Notes in Computer Science 5252, Springer-Verlag, Berlin

Brown J.R (1991) Solving knapsack sharing with general tradeoff functions, Mathematical Programming 51:55–73

Chu P, Beasley J.E (1998) Genetic algorithm for the multidimensional knapsack problem, Journal of Heuristics 4:63–86

Dantzig G.B (1957) Discrete variable extremum problems, Operations Research 5:266–277

Ecker J.G, Shoemaker N.E (1981) Selecting subsets from the set of non-dominated vectors in multiple objective linear programming, SIAM Journal of Control and Optimization 19:505–515

Fayard D, Plateau G (1982) An algorithm for the solution of the 0-1 knapsack problem, Computing 28:269–287

Freville A, Plateau G (1997) The 0-1 bidimensional knapsack problem: toward an efficient high-level primitive tool, Journal of Heuristics 2:147–167

Garey M, Johnson D (1979) Computers and Intractability : a Guide to the Theory of NP-Completness, W.H. Freeman and Company, San Francisco, USA

Gilmore P.C, Gomory R.E (1966) The theory and computation of knapsack functions, Operations Research 13:879–919

Hifi M, Michrafy M and Sbihi A (2006) A Reactive Local Search-Based Algorithm for the Multiple-choice Multidimensional Knapasck Problem, Computational Optimization and Applications 33(2–3):271–285

Hifi M, Sadfi S, Sbihi A (2002) An efficient algorithm for the knapsack sharing problem, Computational Optimization and Applications 23:27–45

Iida H (1999) A note on the max-min 0-1 knapsack problem, Journal of Combinatorial Optimization 3:89–94

Luss H (1992) Minmax resource allocation problems: optimization and parametric analysis, European Journal of Operational Research 60:76–86

Martello S, Toth P (1990) Knapsack Problems: Algorithms and Computer Implementation, John Wiley: New York

Martello S, Pisinger D, Toth P (1999) Dynamic programming and strong bounds for the 0-1 knapsack problem, Management Science 45:414–424

Martello S, Pisinger D, Toth P (2000) New trends in exact algorithms for the 0-1 knapsack problem, European Journal of Operational Research 123:325–332

Nash S.G, Sofer A (1996) Linear and non linear programming, McGraw-Hill International Editions

Pang J.S, Yu C.S (1989) A min-max resource allocation problem with substitutions, European Journal of Operational Research 41:218–223

Pisinger D (1995), A minimal algorithm for the Multiple-choice Knapsack Problem, European Journal of Operational Research 83:394–410

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

Sbihi A (2007) A best-first exact algorithm for the multiple-choice multidimensional knapsack problem, Journal of Combinatorial Optimization 13-4:337–351

Steuer R.E (1986) Multiple criteria optimization: theory, computation and application:Wiley, New York

Tang C.S (1988) A max-min allocation problem: its solutions and applications, Operations Research 36:359–367

Taniguchi F, Yamada T, Kataoka S (2008) Heuristic and exact algorithms for the maxmin optimization of the multi-scenario knapsack problem, Computers and Operations Research 35:2034–2048

Yamada T, Futakawa M, Kataoka S (1998) Some exact algorithms for the knapsack sharing problem, European Journal of Operational Research 106:177–183

Yu G (1996) On the max-min 0-1 knapsack problem with robust optimization applications, Operations Research 44-2:407–415

Appendix: An upper bound computation

Herein we show how to compute UB. First, the MSM^{2}KP is reduced to a single knapsack problem by considering the item j of profit $p_j(\lambda)$ as a convex combination of all the P alternative profits : $p_j(\lambda) := \sum_{\pi=1}^{P} \lambda^\pi v_j^\pi$; $\sum_\pi \lambda^\pi = 1$. Then the MSM^{2}KP relaxation is formulated as :

$$(LP) \begin{cases} \text{Maximize} & Z(X) = \sum_{j \in J} p_j(\lambda) x_j \\ \text{Subject to} & \sum_{j \in J} w_j x_j \leq C \\ & 0 \leq x_j \leq 1, \text{ for } j = 1, \ldots, n \end{cases}$$

To determine the parameters (λ^π), we consider at once an auxiliary problem Aux as a single scenario MSM^{2}KP :

$$(Aux) \begin{cases} \text{Maximize} & Z(X) = \sum_{j \in J} v_j^\pi x_j \\ \text{Subject to} & \sum_{j \in J} w_j x_j \leq C \\ & x_j \in \{0, 1\}, \text{ for } j = 1, \ldots, n \end{cases}$$

Aux denotes a single knapsack problem KP. Its LP relaxation is given by Aux$_{LP}$:

$$(Aux_{LP}) \begin{cases} \text{Maximize} & Z(X) = \sum_{j \in J} v_j^\pi x_j \\ \text{Subject to} & \sum_{j \in J} w_j x_j \leq C \\ & 0 \leq x_j \leq 1, \text{ for } j = 1, \ldots, n \end{cases}$$

It exists a critical item $\ell = \min\{k : \sum_{j=1}^{k} w_j > C\}$ such that $\sum_{j=1}^{\ell-1} w_j \leq C < \sum_{j=1}^{\ell} w_j$. Then the Aux$_{LP}$ optimal solution is given by:

$$x_j := \begin{cases} 1 & \text{if } j \leq \ell - 1 \\ \frac{c - \sum_{k=1}^{\ell-1} w_k}{w_\ell} & \text{if } j = \ell \\ 0 & \text{if } j \geq \ell + 1 \end{cases}$$

And of objective value $Z_{Aux_{LP}}(X^*) = \sum_{j=1}^{\ell-1} v_j^\pi + \frac{c - \sum_{j=1}^{\ell-1} w_j}{w_\ell} v_\ell^\pi$. It represents an upper bound for Aux.

The Dantzig Dantzig (1957) upper bound for Aux is then given by :

$$UB^\pi := \sum_{j=1}^{\ell-1} v_j^\pi + \left\lfloor \frac{c - \sum_{j=1}^{\ell-1} w_j}{w_\ell} v_\ell^\pi \right\rfloor .$$

Then, we consider $\lambda^\pi := \frac{UB^\pi}{\sum_\pi UB^\pi}$, $\forall \pi = 1, \ldots, P$. Knowing these parameters, it is then easy to compute the combined profits $p_j(\lambda) := \sum_{\pi=1}^{P} \left(\frac{UB^\pi}{\sum_\pi UB^\pi} \right) v_j^\pi$. We solve LP and the Dantzig upper bound UB for MSM^2KP is simply the obtained objective value of LP.