Efficient mat heuristic for solving the Multiple-Choice Knapsack Problem with Setup

Yassine Adouani
Laboratory of Modeling and Optimization for Decisional, Industrial and Logistic Systems (MODILS), University of Sfax, Tunisia, adouaniyassine@gmail.com
Malek Masmoudi
University of Sharjah, College of Engineering, Sharjah, UAE, mmasmoudi@sharjah.ac.ae
Bassem Jarboui
Higher Colleges of Technology, UAE, bassem.jarbou@yahoo.fr

Abstract

The knapsack problem with setup (KPS) is a challenging problem in combinatorial optimization. This paper deals with the generalized problem of KPS called multiple-choice knapsack problem with setup (MCKS). The MCKS has several applications in industry such as the assignment of production to plants. A combination of Iterated local search with integer linear programming is provided and enhanced by LP-relaxation to solve the MCKS. The provided mat heuristic is called LP-ILS&IP. A sensitivity analysis is provided to justify the components of the LP-ILS&IP and numerical experiments are conducted on a set of 120 benchmark instances to show the competitiveness of the LP-ILS&IP compared to the best state-of-the-art solving techniques.

Keywords
Knapsack problem with setup, Iterated Local Search, Integer linear programming, LP-relaxation, Mat heuristic

1. Introduction

This paper deals with the multiple-choice knapsack problem with setup (MCKS) (Yang 2006; Adouani et al. 2019). The MCKS has several applications in industry e.g. the assignment of products (families of products) to plants is equivalent to the assignment of items (class of items) to knapsacks in MCKS. In fact, the selection of one plant to produce each item is justified by: the capacity and global budget that limit the production, the procurement of raw materials and logistics operation costs that differ from a plant to another, and the setup and running-in times that is of utmost importance. Moreover, items from the same family can be produced in different plants respecting Jordan and Graves (1995). In addition, to move from one item production to another, a setup is required. We note that the setup between items from the same family is quite negligible e.g. simple setting is required, but the setup from one class of items to another is time consuming and costly e.g. change the molds in the molding machine. The price of an item depends on the plant where it is produced i.e. the profit for an item varies from a plant to another i.e. plant dependent parameter, but the processing time remains the same i.e. plant independent parameter.

The MCKS is a general extension of the classical knapsack problem (KP) (Martello and Toth 1990) where the items belong to classes and an item can be processed in a knapsack only if its class is assigned to this knapsack. The installation of a class requires setup time i.e. resource consumption and cost. MCKS is a generalization of KPS (Chebil and Khemakhem 2015) to multiple knapsacks. It is also a special case of MKPS (Lahyani et al. 2019) while items from the same class can be assigned to different knapsacks. The generalized quadratic multiple knapsack problem (GQMKP) (Avci and Topaloglu 2017) is a special case of MCKS that considers additional profit when two items (j, j') are assigned to the same knapsack.

Approximate and exact methods have been developed for KPS variants. Yang and Bulfin (2009) proposed an exact approach based on a B&B for KPS and showed its limitation for large instances. Yang et al (2009) provided a B&B framework based on the two-stage strategy for the KPS, and particularly analyzed the influence of the correlation between the profit and weight of items on the effectiveness of the framework. Chebil and Khemakhem (2015) proposed a DP algorithm and showed its limitation for large-scale KPS instances (up to 10000 items). For the same problem, Pfersch and Rosario (2018) provided a DP algorithm based on reduction techniques that outperforms the

Approximate and exact methods show drawbacks and advantages, when solving complex problems in general and KPS variants. Therefore, the hybridization of exact and approximate methods has been performed by many researchers to take advantages of both approaches (Dumitrescu and Stützel 2003, Puchinger and Raidl 2005; Jourdan et al. 2009; Blum and Calvo 2015; Hernandez et al. 2019; Turkeş et al. 2021). The matheuristics combining mathematical programming and metaheuristics have particularly received the intensification of many researchers (Hanafi et al. 2010). Burke et al. (2010) provided a matheuristic combining variable neighborhood search (VNS) and integer programming (IP) for Highly Constrained Nurse Rostering Problems. A matheuristic combining Linear Programming (LP) and Tabu Search (TS) to solve the multiple knapsack problem (MKP) is provided by Vasquez and Hao (2001), where the LP is used to solve exactly a relaxation of the problem, and the TS is used to explore the solution neighborhood.

Few works dealt with the MCKS. The related mathematical model is provided in (Yang 2006; Adouani et al. 2019). Due to the complexity of the MCKS (see Section 5), Adouani et al. (2019) provided a matheuristic combining VNS with IP that shows a high performance when tested on a set of 120 instances. Inspired from Adouani et al. 2019, we provide and experiment a new idea of a matheuristic combining LP-relaxation enhancing iterated local search (ILS) technique with IP. The provided matheuristic is called LP-ILS&IP and relies on an effective exploration of the solution space by separating variables into two levels like in Della et al. (2017): The LP-relaxation enhancing ILS technique assign classes to knapsacks and the IP includes items to knapsacks. The numerical results provided in Section 4 show the performance of the LP-ILS&IP approach in comparison to the IP (solved with CPLEX 12.7) and the VNS&IP of Adouani et al. (2019) on the existing benchmark of MCKS. The rest of the article is organized as follows: Section 2 contains the mathematical formulation of MCKS. In Section 3, we detail the provided LP-ILS&IP matheuristic. The experiments and analysis are provided in Section 4. In Section 5, we conclude the work and give future research perspectives.

2. Methods (Problem formulation)

Let consider a set of $M$ knapsacks, a set of $N$ classes of items and a limited budget $b$. Each class $i \in \{1, ..., N\}$ is composed of $n_i$ items. Let $c_{im}$, negative integer number ($c_{im} < 0$), denotes the setup cost of class $i$ in knapsack $m$, and $d_i$, a positive integer number ($d_i > 0$), denotes the setup budget consumption related to class $i$. Each item $j \in \{1, ..., n_i\}$ of a class $i$ has a profit $p_{im}$ when assigned to m ($i \in \{1, ..., N\}, j \in \{1, ..., n_i\}, m \in \{1, ..., M\}$), and a budget consumption $a_{ij}$ ($i \in \{1, ..., N\}, j \in \{1, ..., n_i\}$). For classes and items assignment to knapsacks, we consider two sets of binary decision variables $y_{im}$ and $x_{ijm}$, respectively. The variable $y_{im}$ is equal to 1 if knapsack $m$ includes items belonging to class $i$ and 0 otherwise. The variable $x_{ijm}$ is equal to 1 if item $j$ of class $i$ is included in knapsack $m$, 0 otherwise. The IP model for MCKS is expressed as follows in (Yang 2006; Adouani et al. 2019):

\[
\text{Max } Z = \sum_{m=1}^{M} \sum_{i=1}^{N} (c_{im}y_{im} + \sum_{j=1}^{n_i} p_{ijm}x_{ijm}) \tag{1}
\]

s.t. \[
\sum_{m=1}^{M} \sum_{i=1}^{N} (d_i y_{im} + \sum_{j=1}^{n_i} a_{ij}x_{ijm}) \leq b \tag{2}
\]
\[
x_{ijm} \leq y_{im}; \forall i \in \{1, ..., N\}, \forall j \in \{1, ..., n_i\}, \forall m \in \{1, ..., M\} \tag{3}
\]
The model contains $1+M*S+S$ constraints and $M*N+M*S$ variables, where $S=\sum_{i=1}^{N} n_i$. Objective function (1) maximizes the profit of selected items minus the setup costs of the selected classes for all knapsacks. Constraint (2) guarantees that the sum of the total weight of selected items plus the class setup cost consumption does not exceed the budget $b$. Constraint (3) ensures that each item is selected only if it belongs to a class that has been setup. Constraint (4) guarantees that each item is selected to be realized in one knapsack at most. Constraint (5) guarantees that the decision variables are binary.

The CPLEX 12.7 solver (with default setting) shows its limitation to solve this IP model, especially when considering large instances (see section 4): only 27 instances have been solved to optimality, and the computation for 93 instances terminates with an out of memory or stopped at a limit of computation time equal to $1$ h. This result motivated us to invest in the development of an efficient mat heuristic that we detail in the following section.

3. LP-ILS&IP: efficient Mat heuristic for MCKS

The based idea of our approach is to divide the decision variables into two levels: The first level is about assigning classes to knapsacks by approximately determining the variables $y_{im}$ ($i \in \{1, \ldots, N\}; m \in \{1, \ldots, M\}$) using the LP-relaxation enhancing ILS technique. The second level solves the MCKS[Y] model (objective function 12, and constraints 13-14) to determine the variables $x_{ijm}$ ($i \in \{1, \ldots, N\}; j \in \{1, \ldots, n_i\}; m \in \{1, \ldots, M\}$) using ILP to determine the optimal values of $x_{ijm}$. Note that the found values of $y_{im}$ and $x_{ijm}$ does not necessarily represent an optimal solution of MCKS.

**Algorithm 1.** LP – ILS&IP

<table>
<thead>
<tr>
<th><strong>input:</strong> MCKS data</th>
</tr>
</thead>
<tbody>
<tr>
<td>iter ← 1; iter$_{\text{max}}$ ← $N$;</td>
</tr>
<tr>
<td>S$_0$ ← Construction heuristic RBH</td>
</tr>
<tr>
<td>Best ← S$<em>0$; S$</em>{\text{cur}}$ ← S$_0$;</td>
</tr>
<tr>
<td>While (iter &lt; iter$_{\text{max}}$) do</td>
</tr>
<tr>
<td>(Y, UB) ← Exchange-classes (Best);</td>
</tr>
<tr>
<td>Y ← Perturb-classes (Y, UB);</td>
</tr>
<tr>
<td>S$_{\text{cur}}$ ← Solve MCKS[Y];</td>
</tr>
<tr>
<td>If (S$_{\text{cur}}$ &gt; Best) then</td>
</tr>
<tr>
<td>iter ← 1;</td>
</tr>
<tr>
<td>Best ← S$_{\text{cur}}$;</td>
</tr>
<tr>
<td>Else</td>
</tr>
<tr>
<td>iter ← iter + 1;</td>
</tr>
<tr>
<td>Endif</td>
</tr>
<tr>
<td>Endwhile</td>
</tr>
<tr>
<td>Return Best</td>
</tr>
</tbody>
</table>

First, the LP-ILS&IP begins with a construction method called reduction-based heuristic (called RBH) to obtain an initial solution. Second, the LP based ILS with its two procedures: LP-relaxation enhancing exchange move denoted exchange-classes procedure (EC), and LP-relaxation enhancing perturbation move denoted Perturb-classes procedure (PC) are applied to improve the values of $y_{im}$ and generate a new MCKS[Y]. Third, the MCKS[Y] is solved using CPLEX solver. As the values of $y_{im}$ are not guaranteed to be optimal, we repeat these two latter steps until stopping condition is met. Algorithm1 presents the framework of the LP-ILS&IP approach. The three components of the LP-ILS&IP: construction heuristic RBH, Exchange-classes local search, and Perturb-classes procedure, are explained in the following sections.
3.1 Construction heuristic RBH
The MCKS is a special case of the (G)MKPS while considering only one global budget and multiple knapsacks. To generate a first solution for the MCKS, we were inspired by the RBH of (Adouani et al. 2020) and propose the following procedure composed of three steps:

**First step:** We reduce the size of the MCKS instance so that all the items of each class \( i \in \{1, ..., N\} \) are substituted by a single element named jumbo element, that is identified by a weight \( w_i \) and a profit \( \rho_{im} \) with \( w_i = \sum_{j=1}^{n_i} a_{ij} \) and \( \rho_{im} = \sum_{j=1}^{n_i} p_{ijm} \) \( i \in \{1, ..., N\}; m \in \{1, ..., M\} \). We consider variable \( x_{im} \) equal to 1 if the jumbo element \( i \) is assigned to knapsack \( m \), 0 otherwise. As the weight \( w_i \) of the jumbo element \( i \) can exceed the budget \( b \), the variables \( x_{im} \) are relaxed i.e. \( 0 \leq x_{im} \leq 1 \) and variables \( y_{im} \) remain binary to provide a feasible combination of \( y_{im} \) variables (vector \( Y \)) and to ensure that we will count the total setup cost of a class \( i \) even if we select a fraction of the jumbo item of this class (the same for the setup budget consumption). Thus, the reduced MCKS, named \( \text{MCKS}_{red} \) is mathematically formulated as follows:

\[
\begin{align*}
\text{Max } Z_{red} &= \sum_{m=1}^{M} \sum_{i=1}^{N} (\rho_{im} x_{im} + c_{im} y_{im}) \\
\text{s.t.} & \sum_{m=1}^{M} \sum_{i=1}^{N} (w_i x_{im} + d_i y_{im}) \leq b \\
& x_{im} \leq y_{im} ; \forall i \in \{1, ..., N\}; m \in \{1, ..., M\} \\
& \sum_{m=1}^{M} x_{im} \leq 1 ; i \in \{1, ..., N\} \\
& 0 \leq x_{im} \leq 1, \ y_{im} \in \{0,1\}; \ i \in \{1, ..., N\}; m \in \{1, ..., M\}
\end{align*}
\] (7)

**Second step:** This step consists of optimally solving the \( \text{MCKS}_{red} \) using CPLEX (12.7), which returns a feasible combination of the setup variables \( y_{im} \) denoted by 0-1 vector \( Y_m^i \) \( (m \in \{1, ..., M\}) \). The setup variables are fixed to 1 for selected classes \( (y_{im} = 1) \) and fixed to 0 for free classes \( (y_{im} = 0) \). Let \( Y_m = Y_m^1 \cup Y_m^0 \) with \( Y_m^1 = \{i \in \{1, ..., N\} | y_{im} = 1\} \) and \( Y_m^0 = \{i \in \{1, ..., N\} | y_{im} = 0\} \). We note that the obtained values for \( x_{im} \) are not important in this step and will be optimally solved in the next step.

**Third step:** Once the classes (vector \( Y_m^i \)) are fixed, we determine the \( x_{ijm} \) variables by optimally solving the following MCKS[Y] model:

\[
\begin{align*}
\text{Max } Z &= \theta + \sum_{m=1}^{M} \sum_{i \in Y_m^1} \sum_{j=1}^{n_i} p_{ijm} x_{ijm} \\
\text{s.t.} & \sum_{m=1}^{M} \sum_{i \in Y_m^1} \sum_{j=1}^{n_i} a_{ij} x_{ijm} \leq b' \\
& x_{ijm} \in \{0,1\}; \forall i \in Y_m^1 ; j \in \{1, ..., n_i\}; \forall m \in \{1, ..., M\}
\end{align*}
\] (12)

Where constants \( \theta = \sum_{m=1}^{M} \sum_{i \in Y_m^1} c_{im} \), \( \gamma = \sum_{m=1}^{M} \sum_{i \in Y_m^1} d_i \), and \( b' = b - \gamma \). The \( \text{MCKS}[Y] \) is solved using CPLEX. The initial solution is represented by two vectors of variables: \( Y = \{y_{im}, i = 1, ..., N; m = 1, ..., M\} \) and \( X = \{x_{ijm}, i = 1, ..., N; j = 1, ..., n_i; m = 1, ..., M\} \).

3.2 LP-relaxation enhancing exchange (EC)
The EC local search starts by the solution obtained by solving the MCKS(Y) (denoted \( \text{Best} \)) and consists of efficiently permute classes between different knapsacks i.e. update some values of \( Y \) from 0 to 1 and vice versa. More precisely, we swap the values of \( y_{im} \in Y_m \) and \( y_{jm} \in Y_j \) \( (\forall \ i, j \in \{1, ..., N\}; \forall \ m \in \{1, ..., M - 1\}; \forall \ k \in \{m + 1, ..., M\}) \). Six types of swaps moves (\( N_1 \) to \( N_6 \)) have been considered as neighborhood structures within the EC:
- **N₁**: swap class $i$ from knapsack $m$ with class $j$ from knapsack $k$ i.e. $i \in Y^1_m$ and $j \in Y^1_k$.
- **N₂**: swap class $i$ from knapsack $m$ with two classes $j$ and $j'$ from knapsack $k$ i.e. $i \in Y^1_m$ and $j, j' \in Y^1_k$.
- **N₃**: swap two classes $i$ and $i'$ from knapsack $m$ with class $j$ from knapsack $k$ i.e. $i, i' \in Y^1_m$ and $j \in Y^1_k$.
- **N₄**: swap class $i$ from knapsack $m$ with class $j$ from the set of non-selected classes i.e. $i \in Y^1_m$; $j \in Y_{free}$.
  where, $Y_{free} = \bigcap_{m=1}^{M} \bar{Y}^0_m$ contain the free families.
- **N₅**: swap class $i$ from knapsack $m$ with two classes $j$ and $j'$ from $Y_{free}$ i.e. $i \in Y^1_m$ and $j, j' \in Y_{free}$
- **N₆**: swap two classes $i$ and $i'$ from knapsack $m$ with class $j$ from $Y_{free}$ i.e. $i, i' \in Y^1_m$ and $j \in Y_{free}$.

After each swap, a new combination of $y_{im}$ variables are obtained and hence a new MCKS[Y] model is determined. Let denote the $MCKS[Y]^{LP}$ be the continuous relaxation of $MCKS[Y]$ by relaxing the integrity constraint of $x_{ijm}$ variables (14). To save computational effort, before optimally solving $MCKS[Y]^{LP}$ for combinations of $y_{im}$ variables obtained by iteratively applying the six neighborhood structures. The pseudo code of EC is present in Algorithm 2.

### Algorithm 2: LP-relaxation enhancing exchange

**Input** Best solution for $MCKS$, and $N_k$ (k = {1,...,6})

$UB_{best} \leftarrow$ Best;

While $(k \leq 6)$ do

$Y \leftarrow$ Apply $N_k$ ($UB_{best}$);

$UB_Y \leftarrow$ solve $MCKS[Y]^{LP}$;

If ($UB_Y > UB_{best}$) then

$UB_{best} \leftarrow UB_Y$;

$Y_{best} \leftarrow Y$;

Else

$k \leftarrow k + 1$;

Endif

Endwhile

Return $UB_{best}$ and $Y_{best}$ vector

### 3.3 LP-relaxation enhancing perturbation (PC)

The provided PC strongly perturbs the best solution so that to jump the local optima and give a new starting solution. The PC is composed of two steps:

**Step 1**: Let $N$ be the number of selected classes that can lead to an optimal solution of MCKS. $N$ is bounded by solving two linear continuous problems $MCKS_{min}$ and $MCKS_{max}$. The main idea consists of minimizing and maximizing $NC = \sum_{m=1}^{M} \sum_{i=1}^{N} y_{im}$ (15) with constraints (2-4), new constraint (16) guaranteeing that the total profit must be superior to the best solution found, and constraint (17) of non-integrity. The $MCKS_{min}$ and $MCKS_{max}$ are obtained, respectively:

$$Max \text{ or } Min \quad NC = \sum_{m=1}^{M} \sum_{i=1}^{N} y_{im}$$

$$\sum_{m=1}^{M} \sum_{i=1}^{N} (d_i y_{im} + \sum_{j=1}^{n_i} a_{ij} x_{ijm}) \leq b$$

$$x_{ijm} \leq y_{im}; \forall i \in \{1, ..., N\}, \forall j \in \{1, ..., n_i\}, \forall m \in \{1, ..., M\}$$

$$\sum_{m=1}^{M} x_{ijm} \leq 1; \forall i \in \{1, ..., N\}, \forall j \in \{1, ..., n_i\}$$

$$\sum_{m=1}^{M} \sum_{i=1}^{N} (c_{im} y_{im} + \sum_{j=1}^{n_i} p_{ijm} x_{ijm}) \geq Z (S_{best})$$

© IEOM Society International
\[ x_{ijm}, y_{im} \in [0,1]; \forall i \in \{1, ..., N\}, \forall j \in \{1, ..., n_i\}, \forall m \in \{1, ..., M\} \] (17)

The main idea is to iteratively solve \( MCKS_{\min} \) and \( MCKS_{\max} \) to obtain respectively the minimum and maximum numbers of classes \( N_{\max} \) and \( N_{\min} \). The PC starts by randomly choosing \( N' \) families, \( N' \in [N_{\min}, N_{\max}] \) and randomly affecting the \( N' \) classes to randomly selected knapsacks i.e. randomly fixing \( y_{im} \) variables.

**Step 2:** For each new \( Y \) vector, the \( MCKS[Y]^{LP} \) is solved optimally using ILP i.e. obtaining the upper bound \( UB_Y \) of \( MCKS[Y] \). The new feasible vector \( Y \) is accepted only if \( (UB_Y > UB_{best}) \), while \( UB_{best} \) is the best upper bound founded.

The pseudo code of Perturb-classes procedure is presented in Algorithm 3.

---

**Algorithm 3: LP-relaxation enhancing perturbation**

**Input** \( Y \), and \( UB \), returned by exchange-families

- \( N_{\min} \leftarrow \text{Solve } MCKS_{\min} \);  
- \( N_{\max} \leftarrow \text{Solve } MCKS_{\max} \);  
- \( UB_{best} \leftarrow UB \);  
- \( Y_{best} \leftarrow Y \);

**For** all \( N' \in [N_{\min}, N_{\max}] \) **do**

- **List** \( \leftarrow \text{Randomly choose } N' \text{ classes from } \{1, ..., N\} \);
- \( iter_{max} \leftarrow N' \);
- \( iter \leftarrow 0 \);
- **While** \( (iter < iter_{max}) \)
  - \( Y \leftarrow \text{Randomly assign all } N' \text{ classes to } M \text{ knapsacks} \);  
  - \( UB_Y \leftarrow \text{solve } MCKS[Y]^{LP} \);
  - **If** \( (UB_Y > UB_{best}) \) **then**
    - \( UB_{best} \leftarrow UB_Y \);  
    - \( Y_{best} \leftarrow Y \);
  - **Endif**
  - \( iter \leftarrow iter + 1 \);
- **Endwhile**

**Return** best feasible \( Y_{best} \) vector

---

4. **Computational experiments**

The provided LP-ILS&IP is developed using C programming language and run on a personal computer with 2.4 GHZ intel core 2 duo B960 processor and 4 GB of memory. We used CPLEX 12.7 with default setting as solver. For the experimentation, we considered the benchmark instances from literature (Adouani et al. 2019) with a total number of knapsacks \( M \in \{5, 10, 15, 20\} \), a small budget, a total number of classes \( N \in \{10, 20, 30\} \), and a total number of items \( n_i \) for each class \( i \in \{90, 110\} \). Before the experimentation results, we provide in what follows the performance analysis of the main LP-ILS&IP components: RBH, EC, and PC. We consider three combinations: the RBH alone, the RBH with Exchange-families, and the three components together (RBH, Exchange-classes and Perturb-families). We consider the benchmark set of instances with \( M=\{5, 10, 15, 20\} \). Fig.1 shows the average deviation \( \text{dev}(\%) \) between each combination’s solutions (named combination\(_{sol} \)) and the best know solutions (named best\(_{sol} \)) on each set of instances, where \( \text{dev}(\%) = 100 \times \left( \frac{\text{best}_{sol} - \text{combination}_{sol}}{\text{best}_{sol}} \right) \).
Figure 1 shows the performance of the RBH heuristic on all instances sets with a dev% average of 4.57%. The performance of the RBH decreases when the number of knapsacks increases. We see a considerable improvement of 3.16% on average by adding Exchange-classes to RBH (second combination), and a supplementary improvement of 1.41% by adding Pertub-classes (Third combination).

Table 1 summarizes the results of ILP (solved with CPLEX 12.7), VNS&IP (Adouani et al. 2019) and LP-ILS&IP on the 120 benchmark instances provided in (Adouani et al. 2019). Each line presents the average of 10 instances with the same values of M and N. The first two columns present the instance set (values of M and N), and the next three columns show the numerical results provided by the ILP, the VNS&IP and the ILS-M. For each instance, we consider 10 independent runs and report the average and best solutions of these 10 runs and the average of CPU times of the 10 runs in the following link: https://goo.gl/w44aUs. The notations Sol_best, Sol_avg and CPU report the average of the 10 best solutions found, the average of the 10 average solutions found and the average of the 10 average CPU times (in seconds), respectively. We note that ILP is stopped at a limit of computation time equal to one hour. The columns dev report the standard deviation from the best solution obtained by the algorithm named approachsol (IPsol, or Sol_best) and the best known solution, which is the best among all, named bestsol known: dev (%) = 100 * (bestsol known − approachsol) / bestsol known.

The results in Table 1 show that the LP-ILS&IP outperforms the VNS&IP and ILP (using CPLEX). More precisely the ILP, VNS&IP and LP-ILS&IP obtain solutions with average dev of 0.107%, 0.001% and 0.000% respectively. The detailed results are provided in the following link: https://goo.gl/w44aUs and show that the ILP (using CPLEX) solved only 27/120 instances to optimality in less than one hour CPU time, and for the rest of instances it terminates with an out of memory or exceeds the execution time limit of one hour. On contrary, the LP-ILS&IP finds the best solutions for all instances except 2/120 (instances 5x30x1 and 5x30x7 in detailed results: https://goo.gl/w44aUs) and 92 new best-known solutions, while VNS&IP finds the best solutions for 28/120 instances. In addition, for the instances where the average and the best results are not the same, the average dev between the best and the average results are 0.001% for LP-ILS&IP and 0.014 % for VNS&IP, which proves the robustness of the LP-ILS&IP.
Table 1. Numerical results on the benchmark instances (Adouani et al. 2019)

<table>
<thead>
<tr>
<th>M</th>
<th>N</th>
<th>IP sol</th>
<th>CPU</th>
<th>dev</th>
<th>Sol\textsubscript{best}</th>
<th>Sol\textsubscript{avg}</th>
<th>CPU</th>
<th>dev</th>
<th>Sol\textsubscript{best}</th>
<th>Sol\textsubscript{avg}</th>
<th>CPU</th>
<th>dev</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>10</td>
<td>1772249</td>
<td>1735</td>
<td>0.065</td>
<td>1773409</td>
<td>1773391</td>
<td>6</td>
<td>0.000</td>
<td>1773409</td>
<td>1773409</td>
<td>0.78</td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>3571514</td>
<td>2863</td>
<td>0.062</td>
<td>3573719</td>
<td>3573703</td>
<td>6</td>
<td>0.000</td>
<td>3575734</td>
<td>3573719</td>
<td>0.25</td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>5398429</td>
<td>2267</td>
<td>0.054</td>
<td>5401333</td>
<td>5401300</td>
<td>6</td>
<td>0.000</td>
<td>5401347</td>
<td>5401303</td>
<td>0.88</td>
<td>0.000</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>1795187</td>
<td>2587</td>
<td>0.001</td>
<td>1795188</td>
<td>1795071</td>
<td>11</td>
<td>0.001</td>
<td>1795204</td>
<td>1795201</td>
<td>2.37</td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>3602956</td>
<td>3439</td>
<td>0.003</td>
<td>3603067</td>
<td>3602013</td>
<td>10</td>
<td>0.000</td>
<td>3603078</td>
<td>3603002</td>
<td>1.92</td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>5445060</td>
<td>2937</td>
<td>0.013</td>
<td>5445715</td>
<td>5445708</td>
<td>11</td>
<td>0.000</td>
<td>5445741</td>
<td>5445715</td>
<td>3.05</td>
<td>0.000</td>
</tr>
<tr>
<td>15</td>
<td>10</td>
<td>1793205</td>
<td>2819</td>
<td>0.116</td>
<td>1795262</td>
<td>1795205</td>
<td>15</td>
<td>0.002</td>
<td>1795295</td>
<td>1795279</td>
<td>1.82</td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>3605797</td>
<td>3333</td>
<td>0.312</td>
<td>3617045</td>
<td>3616030</td>
<td>15</td>
<td>0.001</td>
<td>3617065</td>
<td>3617008</td>
<td>4.27</td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>5471310</td>
<td>3255</td>
<td>0.123</td>
<td>5478013</td>
<td>5477604</td>
<td>15</td>
<td>0.000</td>
<td>5478027</td>
<td>5478002</td>
<td>3.68</td>
<td>0.000</td>
</tr>
<tr>
<td>20</td>
<td>10</td>
<td>1793091</td>
<td>2745</td>
<td>0.205</td>
<td>1796768</td>
<td>1796743</td>
<td>20</td>
<td>0.001</td>
<td>1796778</td>
<td>1796743</td>
<td>1.83</td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>3609105</td>
<td>3481</td>
<td>0.177</td>
<td>3615497</td>
<td>3614050</td>
<td>20</td>
<td>0.001</td>
<td>3615516</td>
<td>3615504</td>
<td>2.17</td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>5454676</td>
<td>2961</td>
<td>0.154</td>
<td>5463066</td>
<td>5460108</td>
<td>20</td>
<td>0.000</td>
<td>5463093</td>
<td>5463023</td>
<td>3.05</td>
<td>0.000</td>
</tr>
<tr>
<td>AVG</td>
<td></td>
<td>3609382</td>
<td>2868</td>
<td>0.076</td>
<td>3613174</td>
<td>3612577</td>
<td>13</td>
<td>0.001</td>
<td>3613190</td>
<td>3613159</td>
<td>2.17</td>
<td>0.000</td>
</tr>
</tbody>
</table>

Table 1 also shows the performance of the LP-ILS&IP on benchmark instances in terms of computation time. We notice that LP-ILS&IP is considerably faster on average than VNS&IP and ILP: 2.17 seconds with LP-ILS&IP versus 13 seconds with VNS&IP and 2868 seconds with ILP.

5. Conclusion

In this article, we studied the multiple-choice knapsack problem with setup (MCKS). A new mat heuristic called LP-ILS&IP combining LP-relaxation enhancing ILS with ILP is provided to solve the MCKS. The performance of the LP-ILS&IP components are studied. The LP-ILS&IP is tested on 120 benchmark instances. Numerical results showed the effectiveness and the efficiency of LP-ILS&IP compared to IP (solved with CPLEX 12.7) and the best algorithm in literature (VNS&IP). The LP-ILS&IP particularly reaches optimality for 25 instances and provides best-known solutions for 118/120 instances. For future work, we expect to improve and generalize our LP-ILS&IP algorithm to deal with other variants of KP problems such as generalized knapsack sharing problem (GKSP).

References


© IEOM Society International
Burke, E.K., Li, J., Qu, R., A hybrid model of integer programming and variable neighborhood search for highly
2010.


Chebil, K., Khemakhem, M., A dynamic programming model for the knapsack problem with setup. Computers

Dumitrescu, I., Stützel, T., Combinations of local search and exact algorithms. In: Cagnoni S. et al. (eds): Applications
of evolutionary computation, EvoWorkshops. Lecture Notes in Computer Science, vol. 2611, pp. 211-223
2003.

Della, C.F., Salassa, F., Scatamacchia, R., An exact approach for the 0-1 knapsack problem with setups. Computers

Freville, A., Plateau, G., Heuristics and reduction methods for multiple constraints 0-1 linear programming problems.

Hanafi, S., Lazić, J., Miladenoči, N., Wilbaut, C., New hybrid matheuristics for solving the multidimensional
2010.

Hernandez, F., Gendreau, M., Jabali, O. et al. A., local branching matheuristic for the multi-vehicle routing problem

Hifi, M., Wu, L., Lagrangian heuristic-based neighbourhood search for the multiple-choice multi-dimensional


Khemakhem, M., Chebil, K., A tree search based combination heuristic for the knapsack problem with setup.

Lahyani, R., Chebil, K., Khemakhem, M., Coelho, L.C., Matheuristics for solving the multiple knapsack problem

Martello, S., Toth, P., Knapsack problems. Algorithms and computer implementations. 605 Third Avenue, New York,

Pferschy, U., Rosario, S., Improved dynamic programming and approximation results for the knapsack problem with

Prandtstetter, M., Raidl, G.R., An integer linear programming approach and a hybrid variable neighborhood search

Puchinger, J., Raidl, G.R., Combining meta-heuristics and exact algorithms in combinatorial optimization. In: Mira,
J. and Alvarez, J.R. (Eds.), Artificial Intelligence and Knowledge Engineering Applications. Berlin


Trili, T., Yahyaoui, H., Krichen, S., An iterated variable neighborhood descent hyperheuristic for the quadratic


Yang, Y., Bulfin, R.L., An exact algorithm for the knapsack problem with setup. Int. J. Operational Research, vol. 5,


Biography
Adouani Yassine received his Master in Operations Research and PhD in Operations Research and Decision Making
from the University of Sfax, Faculty of Economic and Management. His research interests include operations research
and scheduling.

Masmoudi Malek is an Associate Professor of Industrial Engineering at the University of Sharjah, Sharjah, in UAE.
He obtained his PhD in Industrial Engineering from the University of Toulouse in France, and the highest qualification

© IEOM Society International
for university professorship (Habilitation) in Industrial Engineering from the University of Jean-Monnet in France. His research interests include industrial and healthcare optimization and management.

Jarboui Bassem is a Full Professor at University of Sfax in Tunisia. He obtained his PhD in quantitative techniques from the University of Sfax and his research interests include metaheuristics and optimization.