LOCAL BRANCHING STRATEGY-BASED METHOD FOR THE KNAPSACK PROBLEM WITH SETUP

Samah Boukhari[1], Isma Dahmani[2] and Mhand Hifi[3]

[1]LaROMaD, USTHB, BP 32 El Alia, 16111 Alger, Algérie
[2]AMCD-RO, USTHB, BP 32, El Alia, 16111 Bab Ezzouar, Alger, Algerie
[3]EPROAD EA4669, UPJV, 7 rue du Moulin Neuf, 80000 Amiens, France

ABSTRACT

In this paper, we propose to solve the knapsack problem with setups by combining mixed linear relaxation and local branching. The problem with setups can be seen as a generalization of 0–1 knapsack problem, where items belong to disjoint classes (or families) and can be selected only if the corresponding class is activated. The selection of a class involves setup costs and resource consumptions thus affecting both the objective function and the capacity constraint. The mixed linear relaxation can be viewed as driving problem, where it is solved by using a special black-box solver while the local branching tries to enhance the solutions provided by adding a series of invalid / valid constraints. The performance of the proposed method is evaluated on benchmark instances of the literature and new large-scale instances. Its provided results are compared to those reached by the Cplex solver and the best methods available in the literature. New results have been reached.

KEYWORDS

Knapsack, Setups, Local Branching, Relaxation

1. INTRODUCTION

The Knapsack Problem with Setup (namely KPS) can be viewed as a more complex variant of the well-known Knapsack Problem (namely KP), where a set of items is considered that is divided into a set of classes. Each class is characterized by both fixed cost and fixed capacity while an item can be selected if the class containing that item is activated. KPS finds its application in many real-world industrial and financial applications, such as order acceptance and production scheduling. An instance of KPS is characterized by a knapsack capacity C and a set $I = \{1,..., m\}$ of disjoint classes associated with items. Each element j belongs to a given class $ti = \{1,...,n_i\}$ (n_i denotes the number of items belonging to the class i, $i \in I$) and the j-th item of the i-th class has a nonnegative profit p_{ij} and a weight w_{ij}. Furthermore, a nonnegative setup cost fi is incurred and a non-negative setup capacity s_i is consumed in case items of class i are selected in the solution. Without loss of generality, we assume that all input parameters have integer values. The goal of the problem is to maximize the difference between the profits related to the selected items and that related to the fixed costs incurred for setting-up classes without violating the knapsack capacity constraint. The studied problem has applications in many areas such as production planning and scheduling (see Chebil and Khemakhem 2015), energy consumption management (see Della Croce, Salassa, and Scatamacchia 2017), and resource allocation (see

Della Croce, Salassa, and Scatamacchia 2017). The KPS has also a important theoretical status because of its generalization to the classical KP.

The rest of the paper is organized as follows. The related work is exposed in section 2. A formal description of the knapsack problem with setup is presented in Section 3.1. Section 3.2 describes the greedy initialization procedure that is used for achieving a starting solution. Section 3.3 discusses the standard local branching used for a general mixed integer programming. Section 3.4 presents the adaptation of the local branching for solving KPS. Finally, Section 4 exposes the experimental part, where the performance of the proposed method is evaluated on two sets of benchmark instances. The first set, containing small and medium-sized instances, is taken from the literature where the provided results are compared to the best solution values published in the literature. The second set contains random generated large-scale instances, where the provided results are compared to those achieved by the Cplex solver.

2. RELATED WORKS

Guignard [10] tackled the setup knapsack by using a Lagrangean decomposition for the setup knapsack problem, where no restrictions on the non-negativity of both setup cost of each class and the profit of each item are considered.

A special case of KSP has been studied by Akinc [1] and Altay et al. [2], where only the setup cost of each class is taken into account (called the fixed charge knapsack problem). In Akinc [1], an exact algorithm has been designed, which is based upon a classical branch-and-bound scheme, while in Altay et al. [2] the authors tackled the case where items can be fractionated by cross decomposition.

Michel et al. [11] addressed an extended multiple-class integer knapsack problem with setup. For that case, the weights associated to items are related to the classes and the total weight is bounded by both lower and upper weight bounds. Different integer linear programming formulations were proposed and an extended branch-and-bound algorithm that is based upon Horowitz and Sahni method was proposed such that nonnegative setup costs were favored.

The knapsack problem with setup has been studied by Chebil and Khemakhem [4] who proposed a dynamic programming procedure, within pseudo-polynomial time complexity. A special converting formulation was considered in order to reduce the size of the storage capacity, which remains quite expensive when using such type of approach.

Chebil and Khemakhem [5] designed a special truncated tree-search for approximately solving KPS. The method applies an avoid duplication technic that consists in reformulating the original problem into a particular integer program. The experimental part showed the effectiveness of that method, especially the effect of the avoiding duplication technic in terms of improving the quality of the provided solutions.

Furini et al. [9] developed linear-time algorithms for optimally solving the continuous relaxation of different integer linear programming formulations of the knapsack with setup. As mentioned in their experimental part, it has been shown that their algorithms outperform both dynamic programming-based approach and blackbox solver.

Della et al. [7] designed an exact method which handles the structure of the formal description of KSP, where the search process explored the partitioning strategy that is based on splitting the decision variables into two levels. A fixation strategy has been applied for reducing the current subproblem to solve while the blackbox solver is applied for solving the reduced subproblem. The experimental part showed that method remains competitive when compared to Chebil and Khemakhem's [4] dynamic programming method.

Pferschy and al. [12] introduced a new dynamic programming approach which performs better than a previous standard dynamic programming-based procedure; that can be considered as a good alternative for an exact resolution when combined with an ILP solver.

Chebil et al. [6] proposed a multilevel matheuristic for solving large-scale problem instances of the knapsack problem with setup and the multiple knapsack version of the problem. The principle of the method is based (i) on reducing the original instance into a special knapsack instance (each class contains one item) and (ii) on solving the continuous relaxation of the induced problem to provide a feasible solution for the original problem. In order to enhance the quality of the solutions reached, a simple tabu list has been incorporated. The experimental part showed that the proposed method remains competitive when its results were compared to those achieved by the state-of-the-art methods available in the literature.

More recently, Amiri [3] proposed a Lagrangean relaxation-based algorithm for solving the knapsack problem with setup. The method follows the standard adaptation of the Lagrangean relaxation, where a series of local optimal solutions are provided by using a descent method. The performance of the method was evaluated on both the standard set of benchmark instances and very large-scale ones (containing till 500 classes and 2 millions of items) and its achieved results were compared to the best bounds available in the literature.

3. MIXED INTEGER AND LOCAL BRANCHING

3.1. The Model

First, let x_{ij} be the decision variable setting equal to 1 if item j of class i is placed in the knapsack, 0 otherwise. Second, let y_i denote the setup decision variable that is equal to 1 if the family i is activated, 0 otherwise. Then, the formal description of SKP (noted PKPS) can be written as follows:

$$\text{Maximize} \quad \sum_{i=1}^{m}\sum_{j=1}^{n_i} p_{ij}x_{ij} - \sum_{i=1}^{m} f_i y_i \qquad (1)$$

$$\text{Subject to} \quad \sum_{i=1}^{m}\sum_{j=1}^{n_i} w_{ij}x_{ij} + \sum_{i=1}^{m} s_i y_i \le C \qquad (2)$$

$$x_{ij} \le y_i, \quad \forall i \in I, \; j = 1,\ldots,n_i \qquad (3)$$

$$x_{ij} \in \{0,1\}, y_i \in \{0,1\}, \quad \forall i \in I, \; j = 1,\ldots,n_i \qquad (4)$$

Where the objective function (1) maximizes the total profit of the selected items minus the fixed costs incurred for setting up the selected classes. The constraint (2) ensures that the weight of selected items in the knapsack, including all setup capacities related to the activated classes, does not exceed the knapsack capacity C. Finally, constraints (3) (representing the precedence constraints) ensures that an item j is selected only with its activated class j.

3.2. A Starting Solution

A starting solution for KPS can be provided by solving a mixed integer relaxation of the original problem P_{SKP}. Let RP_{SKP} denote the problem provided by relaxing all binary variables, i.e., setting $x_{ij} \in [0, 1]$ $\forall$ j = 1,…, n_i, $\forall$ i $\in$ I and $y_i \in [0, 1]$, $\forall$ i $\in$ I such that RP_{SKP}:

$$\text{Maximize} \quad \sum_{i=1}^{m}\sum_{j=1}^{n_i} p_{ij}x_{ij} - \sum_{i=1}^{m} f_i y_i$$

$$\text{Subject to} \quad \sum_{i=1}^{m}\sum_{j=1}^{n_i} w_{ij}x_{ij} + \sum_{i=1}^{m} s_i y_i \le C$$

$$x_{ij} \le y_i, \quad \forall i \in I, \; j = 1,\ldots,n_i$$

$$x_{ij} \in [0,1], y_i \in [0,1], \quad \forall i \in I, \; j = 1,\ldots,n_i$$

To provide a constructive approximate solution, we first proceed to fix step by step the variables y_i to their binary values and to one the decision variables having a fractional value in PR_{SKP}. Second, one can observe that a single knapsack problem can be provided by considering the decision variables x_{ij} whose classes are activated, i.e., $y_i = 1$. Third and last, an optimal solution of the induced knapsack problem built in the second step, which allows a feasible starting solution for P_{SKP}.

The Greedy Constructive Procedure (noted GCP) used, for providing a starting solution, can be described as follows:

- The relaxation RP_{SKP} is optimized by using the simplex method. Then, we collect the current primal solution by setting ($\overline{X}, \overline{Y}$) as the achieved configuration.
- Let Y' denote the binary structure provided by fixing y_i to its current integer value and all the fractional variables y_i to one,
- Let S be the set of indices of the classes fixed to one: $S= \{i \in I \mid y_i=1\}$.Update $C'=C-\sum_{i \in I} s_i$.
- Let P_{KP} be the knapsack problem with capacity C' and the set of elements x_{ij}, i$\in$I. Each element j of class i$\in$S is characterized by its profit p_{ij} and weight w_{ij} . Then, the P_{KP} problem described as follows: P_{KP} :

$$\text{Maximize} \quad \sum_{i \in S}\sum_{j=1}^{n_i} p_{ij}x_{ij}$$

$$\text{Subject to} \quad \sum_{i \in S}\sum_{j=1}^{n_i} w_{ij}x_{ij} \le C'$$

$$x_{ij} \in \{0,1\}, \quad \forall i \in S, \; j = 1,\ldots,n_i$$

- Let X' be the optimal solution of P_{KP}. Then, (X', Y') denotes the (starting) feasible solution of P_{SKP}.

Algorithm 1 describes the main steps of GCP that is used for providing a starting solution and used as the core of the proposed method, as shown in the rest of the paper.

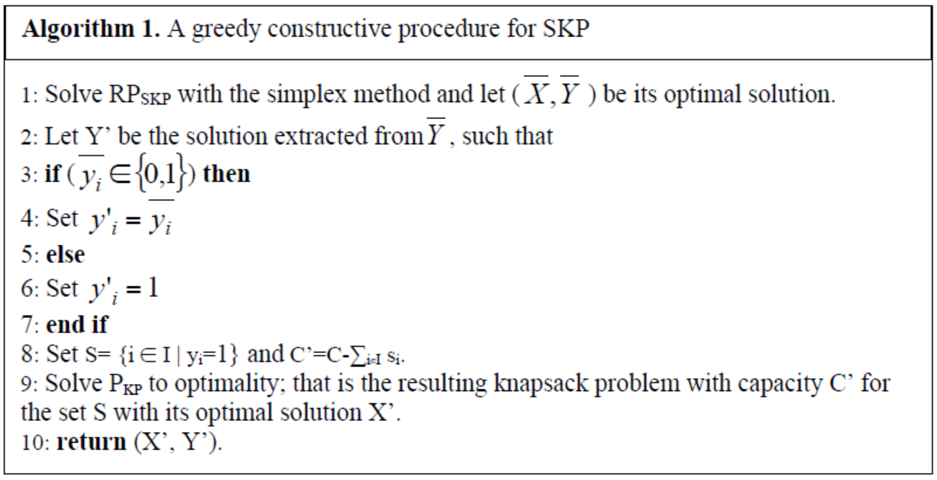
3.3. Local Branching

The original Local Branching (LB) has been first introduced by Fischetti and Lodi [8], especially for efficiently tackling mixed integer formulations of hard combinatorial optimization problems. The solution related to such formulations is often done by calling a black-box solver, like Gurobi, Cplex and Lingo. Because the aforementioned solvers are not able to optimally solve large-scale instances, LB can be used as an alternative at least for enhancing the quality of the solutions. However, since solution procedures using LB-based strategy shown an increasing interest of that method, we propose to solve KPS by using a simple adaptation of LB combined with the greedy constructive procedure GCP (Algorithm 1). The principle of LB-based algorithm may be described as follows:

1. Let S be a starting solution affected to the first node of the developed tree.
2. Initialize the first tree using the solution previously generated.
3. *Iterative phase:*

    i Completely resolve the local tree.
    ii when the local search terminates, the following two cases are distinguished:
        a- A better feasible solution has been provided for the local tree structure. Create a new tree using the aforementioned improved solution like starting solution (called the reference solution).
        b- The solution has not been improved, so abort the local branch.

4. Solve the rest of the search tree.
5. Return the best solution found.


Algorithm 1. A greedy constructive procedure for SKP

```
1: Solve $RP_{SKP}$ with the simplex method and let ($\overline{X}, \overline{Y}$) be its optimal solution.
2: Let Y' be the solution extracted from $\overline{Y}$, such that
3: if ($\overline{y_i} \in \{0,1\}$) then
4: Set $y'_i = \overline{y_i}$
5: else
6: Set $y'_i = 1$
7: end if
8: Set $S = \{i \in I \mid y_i = 1\}$ and $C' = C - \sum_{i \in I} s_i$.
9: Solve $P_{KP}$ to optimality; that is the resulting knapsack problem with capacity C' for
   the set S with its optimal solution X'.
10: return (X', Y').
```

3.4. Adaptation of the Local Branching

In order to adapt LB to the studied problem SKP, we need a starting solution for initializing the first tree, the constraints to use for locally branch on non-searched subspaces and, a blackbox solver for computing the local optimum for each (sub) tree.

Let Y be a feasible reference solution of PKPS provided by the constructive method GCP (cf. Section 3.2). Let S_1 (resp. S_0) be the set related to Y containing elements fixed to one (resp. zero), i.e., $S_1=\{i \mid \forall i\in I,\ y_i=1\}$ (resp. $S_0=\{i \mid \forall i\in I,\ y_i=1\}$). Then, for a given nonnegative integer parameter k, k_{Opt} defines the neighborhood N(Y , k) of the solution Y as the set of the feasible solutions of P_{SKP} satisfying the following additional local branching constraint:

$$\Delta(Y,Y') = \sum_{i\in S_1}(1-y_i) + \sum_{i\in S_0} y_i \le k \qquad (5)$$

where the two terms of left-hand side count the number of binary variables flipping their value (with respect to Y) either from 1 to 0 or from 0 to 1, respectively.

Algorithm 2. Local Branching for KPS

```
1: - Solve PKPS by using the simplex method.
- Let (X'; Y') be the resulting configuration.
- Call GCP (Algorithm 1) for achieving the starting feasible solution $Z^0$.
2: Stop=0;
3: **while** (Stop=0) **do**
4: Solve PKPS with the additional local constraint $\Delta(Y; Y') \le k$, and let $(X^0; Y^0)$ be the provided solution with objective value $Z^0$.
5: **if** ($Z^0 > Z'$) **then**
6: Set $Z' = Z^0$.
7: Remove the old branches.
8: Add the new local branch $\Delta(Y; Y') \ge k + 1$.
9: Set $Y' = Y^0$.
10: **else**
11: Stop=1;
12: **end if**
13: **end while**
14: **return** $Z^0$.
```

Herein, as used in Fischetti and Lodi [8], the local branching constraint is applied as a branching criterion within an enumerative scheme for PKPS. Indeed, given the incumbent solution Y', the solution space associated with the current branching node can be partitioned by separately adding the following disjunction:

$$\Delta(Y,Y') \le k \text{ or } \Delta(Y,Y') \ge k+1 \qquad (6)$$

where *k* denotes a neighborhood-size parameter.

4. COMPUTATIONAL RESULTS

The objective of the computational investigation is to assess the performance of the Local Branching-Based Method (LBBM) by comparing their provided bounds (objective values) to the best known bounds available in the literature. LBBM is evaluated on two sets of instances: the first set, contains 200 small-sized instances, extracted from the literature (cf., Chebil et al. [6]) and a second set, containing 30 large-scale instances, randomly generated following the same generator used by Chebil et al. [6]. Note that the first set (the second set is detailed in section 4.3) includes 20 groups, where each group contains 10 instances.

These instances are considered as the strongly correlated ones, where they are randomly generated applying the following generator described below.

Table 1: Behavior of LBBM on the benchmark instances of the literature: variation of the parameter k

n_i	N	k=2	k=3	k=4	k=5	k=6	k=7	k=8	k=9	k=10
5	500	11004,7	11073,8	11073,8	11073,8	11073,8	11073,8	11073,8	11073,8	11073,8
10		11127,4	11144	11144,6	11144,6	11144,6	11144,6	11144,6	11144,6	11144,6
20		13917,6	13917,6	13917,8	13917,8	13917,8	13917,8	13917,8	13917,8	13917,8
30		13951,8	13952,1	13951,9	13952,2	13952	13952,1	13951,9	13951,7	13952,4
5	1000	19977,8	19977,8	19977,8	19977,8	19977,8	19977,8	19977,8	19977,8	19977,8
10		21943,3	21943,3	21943,3	21943,3	21943,3	21943,3	21943,3	21943,3	21943,3
20		22587,1	22603,5	22629,2	22639,4	22644,1	22648	22648	22648	22648
30		22609	22630,3	22633,5	22647,2	22650,7	22651,1	22650	22650,6	22643,6
5	2500	55185,9	55297,7	55519	55519	55519	55519	55519	55519	55519
10		54840,9	54829,3	54850	54850	54850	54850	54850	54850	54850
20		50128,3	50136,6	50209	50218,7	50245,7	50243,1	50253,4	50244,2	50242,2
30		55445,4	55460,1	55486,5	55514,3	55501,4	55506	55497,1	55505,8	55487,7
5	5000	100302,2	100302,2	100301,9	100302,2	100302,2	100302,2	100302,2	100302,2	100302,2
10		100502,5	100641,9	100645,2	100645,2	100645,2	100649,4	100649,4	100649,4	100649,4
20		100269,8	100544	100630,7	100713,1	100761,1	100774,5	100767,7	100766,9	100761
30		101166,5	101162,2	101183,7	101196,5	101180,8	101177,5	101170,7	101140,4	101152,7
5	10000	221536,1	223128,9	223127,6	223126,4	223126,4	223126,4	223126,4	223126,4	223126,4
10		199177,7	200831,2	201201,8	201227,3	201226,9	201226,9	201227,3	201225,8	201227,9
20		201093	200716,6	201620,8	201522,4	201621,1	201776,9	201776,2	201774,7	201744,8
30		201028,3	200961,6	201486,8	201470,8	201533,3	201520,2	201563,1	201499,7	201485,2
Average		78889,765	79062,735	79176,745	79180,1	79190,86	**79199,03**	**79200,485**	79195,605	79192,49

- The number of classes varies in the discrete interval {10; 20; 30}.
- Number of elements n, related to the number of classes, was fixed to 500, 1000, 2500, 5000 and 10000, respectively.
- The number of items n_i of each class i $\in$I varies in the discrete interval $\left[k-\frac{k}{10}, k+\frac{k}{10}\right]$ where $k=\frac{n}{m}$ and integer.
- Both profits and weights were generated as follows; p_{ij} is randomly generated in the discret interval [10; 100] and $w_{ij} = p_{ij} + 10$, forming strongly correlated instances.
- The setup cost (resp. capacity) of each class i is a random value linking the summation
- The setup cost (resp. capacity) of each class i is a random value linking the summation of the profits (resp. weights) of the items belonging to the class; that is, $f_i = -e_1 \times \sum_{i=1}^{n_i} p_{ij}$ (resp. $s_i = -e_1 \times \sum_{i=1}^{n_i} w_{ij}$), where is drawn from the uniform distribution [0:15, 0:25].
- The knapsack capacity $C = 0.5 \times (\sum_{i\in S} \sum_{j=1}^{n_i} w_{ij})$

The proposed method was coded in C and run on an Intel Pentium Core (TM) i3-6006U with 2GHz.

4.1. Behavior of LBBM on Set 1

In this section, LBBM's behavior is analyzed on the set of instances representing the benchmark instances of the literature. Its achieved results are compared to those reported in Amiri [3]: a Lagrangean relaxation-based heuristic (noted **Lag**), the Two-Phase Reduced Mathheuristic (noted **Mred**) proposed in Chebil et al. [6] and the Cplex solver (noted **Cplex**: that solver was tested using two tunings, where each version was fixed to one hour: (i) automatic search method and, (ii) dynamic search; for each of these versions, the RINS heuristic was fixed to 100); thus, the best objective value achieved by these versions are retuned as the best solution value of Cplex.

Table 2: Performance of LBBM versus Mred and Lag

										Relaxed XY_LB1					
		Cplex			Lag		Mred			*k=7*			*k=8*		
N	n_i	z	cpu	opt	Gap/opt	CPU	Gap /opt	cpu	opt	Gap/opt	CPU	opt	Gap/opt	CPU	opt
5	500	11073,8	222,5408	9	0,16300	5,00	0,00542	0,11	9	0	8,87	10	0	9,54	10
10		11144,6	169,2485	10	0,16900	5,00	0,05563	0,04	4	0	14,28	10	0	12,12	10
20		13917,8	203,6598	8	0,09100	4,00	0,07832	0,01	2	0	40,18	10	0	51,60	10
30		13952,4	329,4308	9	0,03900	4,00	0,06666	0,00	1	0,00215	42,51	9	0,00358	47,51	8
5	1000	19977,8	778,8846	4	0,51400	5,00	0,00200	0,11	9	0	28,94	10	0	29,61	10
10		21943,3	556,6288	0	0,07600	5,00	0,38326	0,04	1	0	41,29	10	0	41,09	10
20		22648	187,573	10	0,24000	5,00	1,02040	0,00	5	0	23,82	10	0	29,18	10
30		22653,6	827,3707	8	0,14600	5,00	0,03311	0,00	0	0,01633	70,61	4	0,02119	78,36	2
5	2500	55519	449,0332	0	0,80500	5,00	0	0,07	10	0	25,78	10	0	25,93	10
10		54850	1648,4682	1	0,24200	5,00	3,53747	0,02	5	0	22,61	10	0	14,78	10
20		50168,2	984,0365	0	0,16200	5,00	0,01731	0,02	7	0,02050	64,12	9	0	87,78	10
30		55512,4	904,0988	0	0,13700	5,00	0,72221	0,01	3	0,03296	80,94	3	0,04899	80,70	1
5	5000	100302,2	1452,1917	0	0,32100	8,00	0	0,01	10	0	0,89	10	0	0,93	10
10		100649,4	859,8139	2	0,31500	8,00	0,00298	0,01	9	0	2,30	10	0	2,07	10
20		100645,8	1274,339	2	0,38200	8,00	0,00079	0,01	9	0,00447	67,23	8	0,01121	83,38	7
30		101200,4	470,8615	8	0,14700	7,00	0,00148	0,01	8	0,02875	80,79	3	0,03547	80,61	0
5	10000	223129,8	1333,462	0	0,94600	17,00	4,25044	0,02	9	0,00152	1,44	7	0,00152	1,43	7
10		201227,9	1961,5044	0	0,30300	16,00	2,25953	0,02	7	0,00050	2,49	9	0,00030	2,43	9
20		200589,1	2742,7227	0	0,31100	16,00	0	0,02	10	0,00292	75,95	8	0,00327	98,74	7
30		201581,8	1382,913	8	0,24700	17,00	0,00015	0,02	9	0,03264	80,83	1	0,01136	80,66	3
Average		**79134,365**	936,939	79	0,28780	7,75	0,62186	0,03	127	**0,00714**	**38,79**	**161**	**0,00684**	**42,92**	**154**

Table 2 reports the results achieved by the four methods tested: Cplex solver, Mred, Lag and LBBM on all instances of set 1. The first two columns of the table show the instance's information (each line corresponds to a group containing 10 instances). Columns from 3 to 5 report the Cplex solver's integer bound (noted z), runtime limit consumed and the number of the optimal solution values matched by the Cplex. Columns 6 and 7 display the average Gap (computed as follows: $Gap = \frac{z_{opt} - z_{procedure}}{z_{opt}} \times 100$) achieved by Lag and its runtime limit (extracted from Amiri [3]. Columns from 8 to 10 tally the average Mred's Gap, its average runtime limit and the number of optimal solution values matched by Mred. Finally, column from 11 to 13 (resp. from 14 to 16) show the LBBM's average Gap with k = 7 (resp. k = 8), its related average runtime and the number of the optimal solution values matched by the algorithm.

In what follows, we comment on the results reported in Table 2, where we compare the number of results obtained by the proposed method LBBM (the solution values) with the best solution values provided by the other three methods. On the one hand, LBBM with k = 7 (resp. k = 8) achieves an average Gap of 0.00714 (resp. 0.00684) while Lag's average Gap is equal to 0.28780 and that of Mred is more greater (0.62186). On the other hand, Cplex matches 79 optimal values over the 200 instances of Set 1 (representing a percentage of 39.5%), Mred provides 127 optimal values (representing a percentage of 63.5%) whereas LBBM matches 161 optimal values for k = 7 (representing a percentage of 80.5%) and 154 ones for k = 8 (representing a percentage of 77%). Finally, even LBBM's average runtime remains higher when compared to those consumed by Lag and Mred, it remains very reasonable for a method using a local branching strategy.

4.2. Behavior of LBBM on the Instances of Set 2

Because Chebil et al. [6]'s instances are note available, we then considered thirty instances representing more largest benchmark instances (these instances are publicly available for other researchers in the domain): the number of classes (m) varies in the discrete interval {50; 150; 300}, the total number of items of each class ($n_{i,}$ $i \in I$) varies in the discrete interval {10000; 100000}, where five instances are considered for each of the six groups. Herein, LBBM's behavior is analyzed on these instances, where its achieved results are also compared to those achieved by the Cplex solver. Table 3 reports the solution values achieved by Cplex solver (its runtime limit was fixed to 3600 seconds) and LBBM on the instances of Set 2. From the table, one can observe what follows:

1. LBBM remains competitive when comparing its results to those achieved by the Cplex solver.
2. LBBM is able to provide ten better average bounds with k = 7 (column 5 in bold-space) than those achieved by Cplex. In this case, LBBM's global average bound is equal to 1472611.17 for k = 7 (resp. 1472487.97 for k = 8) while Cplex's global average bound is equal to 1472310.57.
3. For k = 7, LBBM's average Gap is equal to -0.020 (column 7, last line) and it is equal to -0.012 for k = 8 (column 10, last line) which means that LBBM is capable to improve globally the bounds achieved by the Cplex solver by consuming a smaller runtime (in some cases, it needs only twentieth than the average runtime required by the Cplex).

Table 3: Performance of both Cplex solver and LBBM on large-scale instances.

				k=7			*k=8*		
N	**n_i**	**z_{Cplex}**	**cpu**	**Z_{LBKPS}**	**cpu**	**gap/cplex**	**Z_{LBKPS}**	**cpu**	**gap/cplex**
50	10000	237777	682s	237767,00	162,13	0,004	237739,00	162,31	0,016
		983352	570s	982900,00	161,25	0,046	983257,00	161,17	0,010
		984146	3890,6s	983905,00	161,19	0,024	984035,00	160,81	0,011
		985271	1158,6s	984582,00	241,49	0,070	985104,00	160,69	0,017
		241677	946s	241662,00	161,96	0,006	241558,00	161,36	0,049
150	10000	975600	323s	975180,00	161,21	0,043	975195,00	161,13	0,042
		980232	1h	980138,00	161,16	0,010	980159,00	161,21	0,007
		238051	1h	237943,00	161,30	0,045	237980,00	161,97	0,030
		977670	1h	977538,00	161,08	0,014	977513,00	161,19	0,016
		240515	1323s	240464,00	162,47	0,021	240486,00	161,41	0,012
300	10000	242728	1h	242669,00	161,41	0,024	242634,00	161,11	0,039
		240403	2573s	240352,00	161,56	0,021	240338,00	161,39	0,027
		240297	1h	240245,00	161,47	0,022	240176,00	161,41	0,050
		238 996	1222s	238974,00	161,14	0,009	238968,00	161,46	0,012
		242906	4125s	242882,00	161,74	0,010	242841,00	161,55	0,027
50	100000	2395349	1h	2397736,00	567,01	-0,100	2397579,00	480,00	-0,093
		2409584	1h	2409994,00	1976,91	-0,017	2410086,00	501,18	-0,021
		2399396	1h	2398263,00	565,50	0,047	2399332,00	443,42	0,003
		2399287	1129s	2399167,00	854,53	0,005	2399038,00	531,99	0,010
		2399483	1h	2399205,00	607,82	0,012	2399280,00	432,47	0,008
150	100000	2428789	1h	2428303,00	173,60	0,020	2427565,00	176,91	0,050
		2424999	1h	2427011,00	171,94	-0,083	2426975,00	174,32	-0,081
		2402010	1h	2403642,00	172,28	-0,068	2402408,00	171,96	-0,017
		2432900	1h	2435675,00	176,85	-0,114	2433396,00	170,47	-0,020
		2359761	1h	2360405,00	171,55	-0,027	2360330,00	171,67	-0,024
300	100000	2402071	1h	2402903,00	166,28	-0,035	2402738,00	165,92	-0,028
		2413486	1h	2413577,00	164,30	-0,004	2413521,00	163,99	-0,001
		2419767	1h	2421827,00	164,96	-0,085	2420135,00	165,78	-0,015
		2418549	1h	2417737,00	165,08	0,034	2418532,00	165,19	0,001
		2414265	1h	2415689,00	165,61	-0,059	2415741,00	165,22	-0,061
Average		1472311		1472611,17	292,23	-0,020	1472487,97	216,69	-0,012

5. CONCLUSIONS

In this paper the knapsack problem with setup is studied; that is a more complex variant of the well-known binary knapsack problem. A local branching-based method was proposed for approximately solving the problem. The method combines two future strategies: (i) solving a series of relaxed mixed programs and (ii) adding a series of branching constraints into a developed tree-search. Both strategies cooperate for creating a tree-search, where each path corresponds to adding a series of constraints related to the current or improved solutions. The performance of proposed method was analyzed on two sets of instances containing small and large-scale instances. According to the experimental part, the proposed method remains competitive, where it outperforms methods available in the literature and is able to provide improved solutions for large-scale instances.

REFERENCES

[1] U. Akinc, (2006) "Approximate and exact algorithms for the fixed-charge knapsack problem", European Journal of Operational Research, Vol. 170, No 2, pp 363-375.

[2] N. Altay, JR. Robinson, E. Powell, and K. M. Bretthauer, (2008) "Exact and heuristic solution approaches for the mixed integer setup knapsack problem", European Journal of Operational Research, Vol. 190, No 3, pp 598-609.

[3] A. Amiri, (2019) "A Lagrangean based solution algorithm for the knapsack problem with setups", Expert Systems with Applications, Vol. 143, pp113077.

[4] K. Chebil, and M. Khemakhem, (2015) "A dynamic programming algorithm for the knapsack problem with setup", Computers and Operations Research, Vol. 64, pp 40-50.

[5] K. Chebil and M. Khemakhem, (2016) "A tree search based combination heuristic for the knapsack problem with setup", Computers and Industrial Engineering, Vol. 99, pp 280-286.

[6] K. Chebil, R. Lahyani, M. Khemakhem, and L. C. Coelho, (2019) "Matheuristics for solving the Multiple Knapsack Problem with Setup", *Computers and Industrial Engineering*, Vol. 129, pp 76-89.

[7] C.F. Della, , F. Salassa, and R. Scatamacchia, (2017) "An exact approach for the 0-1 knapsack problem with setups", *Computers and Operations Research*, Vol. 80, pp 61-67.

[8] M. Fischetti and A.Lodi , (2003) "Local branching, Mathematical Programming", Vol. 98, pp. 23{47.

[9] F. Furini, M. Monaci and E. Traversi, (2017) "Exact algorithms for the knapsack problem with setup", *Technical report, Universit Paris Dauphine*.

[10] M. Guignard, (1993) "Solving makespan minimization problems with lagrangean decomposition", Discrete Applied Mathematics, Vol. 42, no 1, pp. 17-29.

[11] S. Michel, N. Perrot, and F. Vanderbeck, (2017) "Knapsack problems with setups", *European Journal of Operational Research*, Vol. 196, no 3, pp. 909-918.

[12] U. Pferschy, R. Scatamacchia, (2018) "Improved dynamic programming and approximation results for the knapsack problem with setups", *International Transactions in Operational Research,* Vol. 25, no 2, pp. 667-682.