An exact approach for the 0–1 knapsack problem with setups

# An exact approach for the 0–1 Knapsack Problem with Setups

Federico Della Croce[a,*], Fabio Salassa[a], Rosario Scatamacchia[a]

[a] *Dipartimento di Automatica e Informatica, Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Torino, Italy*

**Abstract**

We consider the 0–1 Knapsack Problem with Setups. We propose an exact approach which handles the structure of the ILP formulation of the problem. It relies on partitioning the variables set into two levels and exploiting this partitioning. The proposed approach favorably compares to the algorithms in literature and to solver CPLEX 12.5 applied to the ILP formulation. It turns out to be very effective and capable of solving to optimality, within limited CPU time, all instances with up to 100000 variables.

*Keywords:* Knapsack Problem with Setups, Exact approach, 0–1 Programming

## 1. Introduction

The 0–1 Knapsack Problem (KP) is one of the classical problems in combinatorial optimization where a set of items with given profits and weights is available and the aim is to select a subset of the items in order to maximize the total profit without exceeding a known knapsack capacity. KP has been strongly investigated both from a theoretical and a practical point of view (we cite here, among others, two pioneering works [17]-[18], two books [11]-[14] and a comprehensive survey [13]).

The 0–1 Knapsack Problem with Setups (KPS - originally introduced in [6]) can be seen as a generalization of KP where items belong to disjoint families (or classes) and can be selected only if the corresponding family is activated. The selection of a family involves setup costs and resource consumptions thus affecting both the objective function and the capacity constraint. KPS has many applications of interest such as make–to–order production

*Corresponding author:
*URL:* federico.dellacroce@polito.it (Federico Della Croce)

contexts, cargo loading and product category management among others and more generally for resource allocation problems involving classes of elements (see, e.g., [7]). Another application of KPS is originated within the smart-home paradigm where the goal of an efficient management of the buildings energy consumptions is a strong commitment (see Project FLEXMETER funded by the European Commission under H2020 [10]). Here energy providers are requested to manage peak demands while satisfying an aggregated demand curve in order to avoid blackouts due to high peak demands. In this context, it may be required to shut down several home appliances whenever a Demand Response event for overall exceeding energy consumption is identified. This corresponds to select the best appliances to be shut down, by taking into account their relevance and their energy consumption, while the goal is also to minimize the houses involved in this shut down. Here, the families of items are the houses that we do not want to shut down and the items are their appliances. Thus, this is another practical application of KPS.

Several variants of KP have been tackled in the literature. We refer to the work in [12] for a survey on non-standard knapsack problems. In [6], the authors consider the case with setup costs and profits of items being either positive or negative. A pseudo-polynomial time dynamic programming approach and a two-phase enumerative scheme are proposed. Given the pseudo–polynomial time algorithm of [6], and since KPS contains KP as a special case, i.e. when the number of families is equal to 1, KPS is NP–hard in the ordinary sense. In [2], a variant of KPS with fractional items is analyzed and the authors propose both heuristic methods and an exact algorithm based on cross decomposition techniques. In [15], several dynamic programming algorithms have been proposed for the bounded set-up knapsack problem. In [3], algorithms for tackling the so called Fixed Charge Knapsack Problem (FCKP) are presented. FCKP is a special case of KPS without setup capacity consumptions. In [16], a survey on the literature of the KPS variants is provided and a branch and bound scheme is presented.

In [5], a metaheuristic–based algorithm (cross entropy) is introduced to address KPS with more than one copy per item. In [19], a branch and bound algorithm is devised for KPS. This algorithm is capable of tackling instances with up to 10000 variables even though several large correlated instances ran out of memory. The current state-of-the-art exact approach for KPS is the one reported in [7] where an improved dynamic programming procedure is proposed. The procedure favorably compares to the commercial solver CPLEX 12.5 since it solves to optimality instances with up to 10000 items which turn out to be harder than the ones proposed in [19]. Further references can be found in [7].

In this paper we propose an exact approach for KPS relying on an effective exploration of the solution space which exploits the partitioning of the variables set into two levels and requires the solution of several ILP models that show up to be easy to solve in practice. While the idea of approaching a combinatorial optimization problem by solving related simpler ILP formulations was already done in heuristic procedures (see, for instance, [9] for the closest string problem), here we do it within an exact approach. The proposed approach is capable of solving to optimality, in limited time, all instances with up to 100000 variables. The method strongly outperforms both the state-of-the-art approach proposed in [7] and the solver CPLEX 12.5 applied to the standard ILP formulation of KPS. The paper is organized as follows. In Section 2, the ILP formulation of the problem is described. We present the exact approach in Section 3. In Section 4 computational results are discussed. Section 5 concludes the paper with final remarks.

## 2. Notation and problem formulation

KPS can be described as follows. A set of $N$ families of items is given together with a knapsack with capacity $b$. Each family $i \in \{1...N\}$ is composed of $n_i$ items and characterized by a non-negative integer $f_i$ that represents the family setup cost and a non-negative integer $d_i$ that represents the family setup capacity consumption, respectively. Each item $j \in \{1..n_i\}$ of a family $i$ has a non-negative integer profit $p_{ij}$ and a non-negative integer capacity consumption $w_{ij}$. The goal is to maximize the total profit of the selected items minus the fixed costs incurred for setting-up the selected families without exceeding the knapsack capacity $b$.

Let us associate with each item $j$ of family $i$ a binary variable $x_{ij}$ such that $x_{ij} = 1$ if item $j$ of family $i$ is placed in the knapsack, else $x_{ij} = 0$. Also, let us associate with each family $i$ a binary variable $y_i$ such that $y_i = 1$ if the knapsack is setup to accept items belonging to family $i$, else $y_i = 0$. The following ILP formulation of KPS (denoted $KPS_1$) holds.

$\boldsymbol{KPS_1}$:

$$\text{maximize} \quad \sum_{i=1}^{N}\sum_{j=1}^{n_i} p_{ij}x_{ij} - \sum_{i=1}^{N} f_i y_i \tag{1}$$

$$\text{subject to} \quad \sum_{i=1}^{N}\sum_{j=1}^{n_i} w_{ij}x_{ij} + \sum_{i=1}^{N} d_i y_i \leq b \tag{2}$$

$$x_{ij} \leq y_i \quad \forall\, j = 1,\ldots,n_i, \quad \forall\, i = 1,\ldots,N \tag{3}$$

$$x_{ij} \in \{0,1\} \quad \forall\, j = 1,\ldots,n_i, \quad \forall\, i = 1,\ldots,N \tag{4}$$

$$y_i \in \{0,1\} \quad \forall\, i = 1,\ldots,N \tag{5}$$

Here, the objective function (1) maximizes the sum of the profits of the selected items minus the costs induced by the selected families; the capacity constraint (2) guarantees that the sum of weights for the selected items and families does not exceed the capacity value $b$; constraints (3) ensure that an item can be chosen if and only if the corresponding family is activated; finally constraints (4,5) indicate that all variables are binary.

## 3. An exact solution approach

### *3.1. Rationale and preliminaries*

Let denote by $KPS^{LP}$ the continuous relaxation of $KPS_1$. It is known [19] that there exists at least one optimal solution of $KPS^{LP}$ where there is at most one fractional variable $y_i$ while there are typically many fractional variables $x_{ij}$. As an example, we tested an instance from [7] with 10000 variables and 30 families: the optimal continuous solution has one fractional variable $y_i$ and 330 fractional variables $x_{ij}$. Then, a branch on any fractional $x_{ij}$ always induced continuous solutions with more than 300 fractional $x_j$ and 1 fractional $y_i$ (often different from the one related to the original problem). Besides, branching on the fractional $y_i$, induced again fractional continuous solutions (always more than 300 fractional $x_{ij}$ and another fractional $y_i$). This, presumably, is the main reason for which a standard ILP solver runs already into difficulties on several instances of $KPS_1$ with 1000 jobs (see Section 4). Our approach instead aims to exploit the structure of KPS, where the set of variables is partitioned into two levels, variables $y_i$ (first level variables) and variables $x_{ij}$ (second level variables). The practical hardness of the problem comes from these two sets of variables that must be properly combined to reach an optimal solution. At the same time, once the families are chosen, KPS boils down to a standard KP. Even if KP is known to be weakly NP–Hard, in practice it is well handled by nowadays ILP solvers (for a comprehensive survey, see [11],[13], and [14]). Notice that, the idea of using approaches based on the repeated solution of NP–hard subproblems is not new. For instance, in [1], the famous shifting bottleneck procedure for the job shop problem was based on the repeated solution of a single machine problem with release times and tails that, although being NP–hard in the strong sense, is well solved in practice by the exact algorithm in [4]. Here, as the selection of the families induces problems that are tractable in practice, we focus on an efficient exploration of the solution space defined by the first level variables.

In particular, we propose an exact approach based on the idea of limiting the range on the number of families that may lead to an optimal solution and seek for solutions within this range. Three main steps are involved. In the

first step an initial feasible solution is computed and a standard variable fixing procedure is applied through the reduced costs of the non–basic variables in the optimal solution of the continuous relaxation of the problem. The second step concerns the detection of the range of possible optimal number of families. This leads to the identification of sub–problems that are tackled in the third phase. We use the ILP solver (CPLEX 12.5) whenever the solution of an ILP model is required by our approach. In the following subsections we describe the three steps of the approach whose pseudo code is presented in Algorithm 1.

*3.2. Initial feasible solution computation and variables fixing*

We start by considering $KPS^{LP}$ where, in addition, we require the sum of the selected families to be integer. Thus, we get the following model (denoted by $KPS_2$).

$\boldsymbol{KPS_2}$:

$$\text{maximize} \quad \sum_{i=1}^{N}\sum_{j=1}^{n_i} p_{ij}x_{ij} - \sum_{i=1}^{N} f_i y_i \tag{6}$$

$$\text{subject to} \quad (2), (3)$$

$$\sum_{i=1}^{N} y_i = k \tag{7}$$

$$0 \leq x_{ij} \leq 1 \quad \forall\, j = 1, \ldots, n_i, \quad \forall\, i = 1, \ldots, N \tag{8}$$

$$0 \leq y_i \leq 1 \quad \forall\, i = 1, \ldots, N \tag{9}$$

$$k \in \mathcal{N} \tag{10}$$

Here, the integrality constraints on variables $x_{ij}$ and $y_i$ of $KPS_1$ are replaced by the inclusion in [0,1] while constraint (7) forces the sum of the families to take an integer value through the integer variable $k$. The optimal solution of this problem gives an upper bound on the KPS optimum. Moreover, the optimal value of $k$, denoted by $k^*$, provides a first guess on the total number of families to include in a solution. Then, we consider again model $KPS_1$ with the additional constraint that the number of families to activate is fixed to a value $S$ and we remove the integrality constraints on variables $x_{ij}$ only. Correspondingly, we get hereafter the following model (denoted by $KPS_3$).

***KPS*$_3$**:

$$\text{maximize} \quad \sum_{i=1}^{N}\sum_{j=1}^{n_i} p_{ij}x_{ij} - \sum_{i=1}^{N} f_i y_i \tag{11}$$

$$\text{subject to} \quad (2), (3)$$

$$\sum_{i=1}^{N} y_i = S \tag{12}$$

$$0 \leq x_{ij} \leq 1 \quad \forall\, j = 1, \ldots, n_i \quad \forall\, i = 1, \ldots, N \tag{13}$$

$$y_i \in \{0, 1\} \quad \forall\, i = 1, \ldots, N \tag{14}$$

We may expect that problem $KPS_3$ is easy to solve as only the $y_i$ variables are binary and the number of families is relatively limited. Further, the solution space is restricted to the hyperplane representing the sum $S$ in constraint (12). This argument shows up to hold in practice. We first solve $KPS_3$ by setting $S = k^*$. The optimal solution provides a feasible combination of $y_i$, denoted by the 0–1 vector $y'$.
If we consider the combination $y'$ in $KPS_1$, we induce a KP with the capacity constraint and objective function modified according to the setups of the families. For the sake of simplicity, hereafter we refer to

$$KPS_1(y') = KPS_1 \cap (y_i = y'_i) \quad \forall\, i = 1, \ldots, N \tag{15}$$

as the standard knapsack problem related to any specific combination of families encoded by vector $y'$.
Solving $KPS_1(y')$ provides a first feasible solution for KPS. Let us denote this solution by $LB' = z_{opt}(KPS_1(y'))$. This is sketched in lines 2-6 of Algorithm 1.
Then, we solve $KPS^{LP}$. We denote the optimal value of $KPS^{LP}$ by $z_{opt}(KPS^{LP})$ and the optimal values of variables $x_{ij}$ and $y_i$ by $x_{ij}^{LP}$ and $y_i^{LP}$ respectively. Let $r_{x_{ij}}$ and $r_{y_i}$ be the reduced costs of non basic variables in the optimal solution of $KPS^{LP}$. We then apply standard variable-fixing techniques from Integer Linear Programming. It is well known (see, for instance, [8]) that, if the gap between the best feasible solution available and the optimal solution value of the continuous relaxation solution is not greater than the absolute value of a non basic variable reduced cost, then the related variable can be fixed to its value in the continuous relaxation solution. Correspondingly, we evaluate the reduced costs of all non basic variables in the optimal solution of $KPS^{LP}$. Then, the following constraints are added to the models (lines 7–8 of Algorithm 1):

$$\forall\, i,j : |r_{x_{ij}}| \geq z_{opt}(KPS^{LP}) - LB', \quad x_{ij} = x_{ij}^{LP} \tag{16}$$
$$\forall\, i : |r_{y_i}| \geq z_{opt}(KPS^{LP}) - LB', \quad y_i = y_i^{LP} \tag{17}$$

*3.3. Identifying the relevant sums of the families*

Given the first solution $LB'$, the number of families in an optimal solution can be bounded straightforwardly by solving two continuous problems. More precisely, we minimize and maximize $\sum y_i$ subject to constraints (2), (3) and to an additional constraint ensuring that the total profit must be strictly greater than the current solution value. The corresponding ILP formulations (denoted by $KPS_{min}$ and $KPS_{max}$ respectively) are as follows.

***KPS*$_{min}$ (*KPS*$_{max}$)**:

$$\min\ (\max) \sum_{i=1}^{N} y_i \tag{18}$$
$$\text{subject to} \quad (2), (3)$$
$$\sum_{i=1}^{N}\sum_{j=1}^{n_i} p_{ij}x_{ij} - \sum_{i=1}^{N} f_i y_i \geq LB' + 1 \tag{19}$$
$$0 \leq x_{ij} \leq 1 \quad \forall\, j = 1,\ldots,n_i, \quad \forall\, i = 1,\ldots,N \tag{20}$$
$$0 \leq y_i \leq 1 \quad \forall\, i = 1,\ldots,N \tag{21}$$

Ceiling and flooring the optimal solution values of the above problems yield $S_{min} = \lceil z_{opt}(KPS_{min}) \rceil$ and $S_{max} = \lfloor z_{opt}(KPS_{max}) \rfloor$, namely the lower and upper bound on the number of families possibly leading to an optimal solution of KPS. The second step of our approach is summarized in lines 9–12 of Algorithm 1.

*3.4. Solving sub–problems*

The third step consists in exploring sub–problems for the possible values of S in the range $[S_{min}, S_{max}]$ (for–loop in lines 13–24 of Algorithm 1).

For each sub–problem we first solve $KPS_3$ and find a combination of families $\bar{y}$ as in subsection *3.2* (lines 14–15 of Algorithm 1). Then we solve $KPS_1(\bar{y})$ and if its optimal value is greater than the current best feasible solution value, we update the latter one (lines 17–20 of Algorithm 1). We solve to optimality a KP, but indeed $\bar{y}$ is not guaranteed to be optimal for $KPS_1$. So we search for another possible combination of $y_i$ within the sub–problem by adding to $KPS_3$ the constraint

$$\sum_{i=1}^{N} \bar{y}_i y_i \leq S - 1 \tag{22}$$

This is a cut in the solution space imposing that at least one of the families of the previous combination must be discarded. We solve $KPS_3$ with one more constraint and apply the same procedure until the upper bound provided by solving $KPS_3$ is not better than the current best solution value or the problem becomes infeasible (while–loop in lines 16–23 of Algorithm 1). We note that $KPS_3$ can turn out to be difficult to solve when further constraints on variables $y_i$ are added. Nevertheless, additional cuts showed up to be reasonably limited. Once all sub–problems have been investigated, an optimal solution of KPS is obtained.

**Algorithm 1** Exact solution approach

1: **Input:** KPS instance.
▷ Find first solution and fix variables
2: $k^* \leftarrow$ solve $KPS_2$;
3: $S \leftarrow k^*$;
4: $(UB', y') \leftarrow$ solve $KPS_3$;
5: $LB' \leftarrow$ solve $KPS_1(y')$;
6: $Best = LB'$;
7: Solve $KPS^{LP}$;
8: Apply (16, 17) and fix variables;
9: $z_{min} \leftarrow$ solve $KPS_{min}$; ▷ Identify the range of families
10: $z_{max} \leftarrow$ solve $KPS_{max}$;
11: $S_{min} = \lceil z_{min} \rceil$;
12: $S_{max} = \lfloor z_{max} \rfloor$;
13: **for** all $s$ in $[S_{min}, S_{max}]$ **do** ▷ Solve sub–problems
14: $S = s$;
15: $(\overline{UB}, \bar{y}) \leftarrow$ solve $KPS_3$;
16: **while** $\overline{UB} \geq Best + 1$ **do**
17: $\overline{LB} \leftarrow$ solve $KPS_1(\bar{y})$;
18: **if** $\overline{LB} > Best$ **then**
19: $Best = \overline{LB}$;
20: **end if**
21: add $(\sum_{i=1}^{N} \bar{y}_i y_i \leq S - 1)$ to $KPS_3$;
22: $(\overline{UB}, \bar{y}) \leftarrow$ solve $KPS_3$;
23: **end while**
24: **end for**
25: **return** $Best$;

We note that steps 1–12 in Algorithm 1 are executed only once, requiring the solution of problems $KPS_2$, $KPS_3$, $KPS_1(y')$, $KPS^{LP}$, $KPS_{min}$ and

$KPS_{max}$ and the variable fixing (running in $O(\sum_{i=1}^{N} n_i)$ time) induced by constraints (16, 17). Also, the for–loop in lines 13–24 is repeated $[S_{max} - S_{min} + 1] = O(N)$ times where at each iteration: (1) $KPS_3$ is solved once and (2) the while-loop in lines 16–23 requiring first the solution of $KPS_1(\bar{y})$ and then the solution of $KPS_3$ until $\overline{UB} \leq Best$. Thus the bottleneck of the algorithm is indeed the total number of times the while–loop is executed which could be potentially large but computational testing indicates that this number is very small in practice (never superior to 33 for instances with up to 100000 items).

## 4. Computational Results

All tests were conducted on an Intel i5 CPU @ 3.3 GHz with 4 GB of RAM. We used the ILP solver CPLEX 12.5 and the code was implemented in the C++ programming language. We generated the instances according to the scheme provided in [19]. In addition, we also considered the instances available in [7].

In the scheme provided in [19], the number of families $N$ is 50 and 100. The cardinalities $n_i$ of the families are integers uniformly distributed in the ranges [40, 60] and [90, 110]. Setup costs and weights are given by

$$f_i = e_1 \left( \sum_{j=1}^{n_i} p_{ij} \right) \tag{23}$$

$$d_i = e_2 \left( \sum_{j=1}^{n_i} w_{ij} \right) \tag{24}$$

where $e_1$ and $e_2$ are uniformly distributed in the intervals [0.05, 0.15], [0.15, 0.25], [0.25, 0.35] and [0.35, 0.45]. In the uncorrelated instances, both the items weights $w_{ij}$ and profits $p_{ij}$ are integer randomly distributed in the range [10, 10000]. In the correlated instances the profits are integer randomly distributed in the range[$w_{ij}$-1000, $w_{ij}$+1000], but if the profits are less than 10, then they range in the interval [10, 100]. The capacity $b$ is an integer randomly distributed in the range $\left[ 0.4 \left( \sum_{i=1}^{N} \sum_{j=1}^{n_i} w_{ij} \right), 0.6 \left( \sum_{i=1}^{N} \sum_{j=1}^{n_i} w_{ij} \right) \right]$.

| | | | CPLEX 12.5 | | | Exact approach | | | |
|---|---|---|---|---|---|---|---|---|---|
| $N$ | $n_i$ | *Setup* | Average time (s) | Max time (s) | #Opt | Average time (s) | Max time (s) | Max #Sub–pb | #Opt |
| 50 | [40-60] | [0.05-0.15] | 0.31 | 0.38 | 10 | 0.50 | 0.64 | 1 | 10 |
| | | [0.15-0.25] | 0.34 | 0.42 | 10 | 0.53 | 0.62 | 1 | 10 |
| | | [0.25-0.35] | 0.45 | 0.59 | 10 | 0.57 | 0.62 | 1 | 10 |
| | | [0.35-0.45] | 0.35 | 0.56 | 10 | 0.45 | 0.57 | 1 | 10 |
| 50 | [90-110] | [0.05-0.15] | 0.49 | 0.66 | 10 | 0.71 | 0.91 | 1 | 10 |
| | | [0.15-0.25] | 0.73 | 0.97 | 10 | 0.90 | 1.03 | 1 | 10 |
| | | [0.25-0.35] | 2.15 | 10.41 | 10 | 0.96 | 1.15 | 1 | 10 |
| | | [0.35-0.45] | 1.12 | 2.68 | 10 | 0.86 | 1.26 | 1 | 10 |
| 100 | [40-60] | [0.05-0.15] | 0.45 | 0.66 | 10 | 0.67 | 0.88 | 1 | 10 |
| | | [0.15-0.25] | 0.69 | 0.91 | 10 | 0.82 | 1.00 | 1 | 10 |
| | | [0.25-0.35] | 0.65 | 1.03 | 10 | 0.80 | 1.17 | 1 | 10 |
| | | [0.35-0.45] | 0.65 | 0.89 | 10 | 0.76 | 0.97 | 1 | 10 |
| 100 | [90-110] | [0.05-0.15] | 0.98 | 1.33 | 10 | 1.16 | 1.58 | 1 | 10 |
| | | [0.15-0.25] | 1.86 | 3.25 | 10 | 1.73 | 2.22 | 1 | 10 |
| | | [0.25-0.35] | 1.35 | 2.15 | 10 | 1.49 | 1.75 | 1 | 10 |
| | | [0.35-0.45] | 1.33 | 2.08 | 10 | 1.52 | 2.41 | 1 | 10 |

Table 1: KPS uncorrelated instances with $w_{ij}$ and $p_{ij}$ in $[10, 10000]$: time (s) and number of optima.

| | | | CPLEX 12.5 | | | Exact approach | | | |
|---|---|---|---|---|---|---|---|---|---|
| $N$ | $n_i$ | *Setup* | Average time (s) | Max time (s) | #Opt | Average time (s) | Max time (s) | Max #Sub–pb | #Opt |
| 50 | [40-60] | [0.05-0.15] | 1.02 | 2.53 | 10 | 0.72 | 1.14 | 2 | 10 |
| | | [0.15-0.25] | 0.67 | 0.97 | 10 | 0.59 | 0.89 | 2 | 10 |
| | | [0.25-0.35] | 0.63 | 1.45 | 10 | 0.61 | 0.78 | 1 | 10 |
| | | [0.35-0.45] | 0.96 | 2.62 | 10 | 0.65 | 0.81 | 2 | 10 |
| 50 | [90-110] | [0.05-0.15] | 3.53 | 13.14 | 10 | 0.99 | 1.31 | 1 | 10 |
| | | [0.15-0.25] | 7.65 | 22.17 | 10 | 1.18 | 1.47 | 1 | 10 |
| | | [0.25-0.35] | 3.41 | 10.87 | 10 | 1.15 | 1.42 | 1 | 10 |
| | | [0.35-0.45] | 9.46 | 39.08 | 10 | 1.27 | 1.81 | 1 | 10 |
| 100 | [40-60] | [0.05-0.15] | 1.51 | 5.40 | 10 | 0.81 | 1.08 | 1 | 10 |
| | | [0.15-0.25] | 1.95 | 7.13 | 10 | 0.98 | 1.31 | 1 | 10 |
| | | [0.25-0.35] | 1.01 | 2.48 | 10 | 1.04 | 1.37 | 2 | 10 |
| | | [0.35-0.45] | 1.37 | 2.50 | 10 | 1.17 | 1.76 | 2 | 10 |
| 100 | [90-110] | [0.05-0.15] | 11.15 | 71.98 | 10 | 1.93 | 2.59 | 1 | 10 |
| | | [0.15-0.25] | 31.14 | 173.21 | 10 | 2.00 | 2.50 | 1 | 10 |
| | | [0.25-0.35] | 71.22 | 652.02 | 10 | 2.30 | 4.57 | 2 | 10 |
| | | [0.35-0.45] | 15.32 | 42.56 | 10 | 2.18 | 3.40 | 2 | 10 |

Table 2: KPS correlated instances with $w_{ij}$ in $[10, 10000]$ and $p_{ij}$ in $[w_{ij}-1000, w_{ij}+1000]$: time (s) and number of optima.

We compared the solutions reached by CPLEX 12.5 running on $KPS_1$ to the solutions obtained with our approach over 10 instances within each category. The results are reported in Tables 1 and 2 in terms of average and maximum CPU time and number of optima reached within a time limit of 1200 seconds. We also report the maximum number of the relevant sub–problems, that is $S_{max} - S_{min} + 1$, identified by our approach.

Uncorrelated instances show up to be very easy to solve for both our approach and CPLEX 12.5. We remark that the same conclusion applies to the instances in [6], which are uncorrelated with positive or negative profits

for the items and setup costs. As mentioned in [19], these instances are not difficult, since a preprocessing step reduces the problems size considerably.

For the correlated instances, CPLEX 12.5 solves to optimality all the instances but performs slightly worse. Our exact approach reaches the optimum over all instances in no more than 5 seconds. We note that the method proposed in [19] requires significantly higher computational time and runs out-of-memory in several cases for similar correlated instances. So, even if we were not able to obtain from the authors the instances reported in [19], we can reasonably expect our method to outperform their approach.

We further tested a stronger correlation between the profits of the items and their weights. More precisely, we generated instances where $w_{ij}$ is an integer uniformly distributed in the range [10, 100], while the profits of items are $p_{ij} = w_{ij} + 10$. The results are provided in Table 3.

| | | | CPLEX 12.5 | | | Exact approach | | | |
|---|---|---|---|---|---|---|---|---|---|
| $N$ | $n_i$ | *Setup* | Average time (s) | Max time (s) | #Opt | Average time (s) | Max time (s) | Max #Sub–pb | #Opt |
| 50 | [40-60] | [0.05-0.15] | 126.17 | 1200.00 | 9 | 1.80 | 4.91 | 2 | 10 |
| | | [0.15-0.25] | 6.76 | 34.10 | 10 | 1.41 | 3.46 | 2 | 10 |
| | | [0.25-0.35] | 4.25 | 10.06 | 10 | 1.20 | 2.43 | 3 | 10 |
| | | [0.35-0.45] | 2.47 | 6.91 | 10 | 0.83 | 1.68 | 2 | 10 |
| 50 | [90-110] | [0.05-0.15] | 701.84 | 1200.00 | 5 | 2.13 | 4.52 | 2 | 10 |
| | | [0.15-0.25] | 241.20 | 1200.00 | 9 | 2.70 | 9.86 | 2 | 10 |
| | | [0.25-0.35] | 307.96 | 1200.00 | 9 | 2.10 | 4.84 | 2 | 10 |
| | | [0.35-0.45] | 28.75 | 214.94 | 10 | 1.49 | 2.14 | 1 | 10 |
| 100 | [40-60] | [0.05-0.15] | 30.66 | 157.59 | 10 | 9.48 | 65.13 | 2 | 10 |
| | | [0.15-0.25] | 18.85 | 100.34 | 10 | 3.83 | 13.62 | 2 | 10 |
| | | [0.25-0.35] | 5.55 | 14.49 | 10 | 2.29 | 5.29 | 2 | 10 |
| | | [0.35-0.45] | 9.65 | 24.24 | 10 | 2.69 | 7.00 | 3 | 10 |
| 100 | [90-110] | [0.05-0.15] | 498.44 | 1200.00 | 7 | 5.56 | 11.92 | 2 | 10 |
| | | [0.15-0.25] | 197.00 | 1200.00 | 9 | 5.35 | 10.66 | 2 | 10 |
| | | [0.25-0.35] | 267.26 | 1200.00 | 9 | 6.40 | 22.07 | 2 | 10 |
| | | [0.35-0.45] | 188.75 | 1200.00 | 9 | 5.37 | 9.95 | 2 | 10 |

Table 3: KPS correlated instances with $w_{ij}$ in $[10, 100]$ and $p_{ij} = w_{ij} + 10$: time (s) and number of optima.

These instances turned out to be harder to solve than the correlated instances in Table 2. A reasonable interpretation is that in [19] a weaker correlation is considered and weights vary in a much wider range ([10, 10000]), increasing the probability of having items much more profitable than others. Nevertheless our approach still manages to handle all instances in very reasonable computational time, while CPLEX 12.5 is not capable of reaching all the optima. It is quite evident from our testing that one of the strengths of our approach is the capacity of drastically limiting the number of subproblems to be explored in the last step of the algorithm. A natural question that may arise is whether this last task can be accomplished just by letting an ILP solver tackle the sub-problems. It would indicate to what extent the

procedure devised in the third step of our method provides an effective contribution in solving the problem. We investigated this aspect by exploring the behaviour of the approach if CPLEX 12.5 is launched (with a time limit of 1200 seconds) on each of the subproblems in the third step of the method, that is the subroblems of subsection 3.4. We denote as *Exact approach (II)* this last version of the proposed approach.

We then compared the two versions of the proposed approach to the dynamic programming algorithm in [7] and to CPLEX 12.5 over a set of instances proposed in [7]. These instances involve a high level of correlation between profits and weights where $w_{ij}$ is an integer uniformly distributed in the range [10,100] and $p_{ij} = w_{ij} + 10$. In Table 4, we report the performances of CPLEX 12.5, the two versions of our approach and the dynamic programming algorithm proposed in [7]. The number of families varies from 5 to 30 and the total number of items $n$ from 500 to 10000. Within each category, 10 instances were tested.

| | | CPLEX 12.5 | | | Exact approach | | | | Dynamic Progr. from [7] | | | Exact approach (II) using CPLEX 12.5 for solving subproblems | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $N$ | $n$ | Average time (s) | Max time (s) | #Opt | Average time (s) | Max time (s) | Max #Sub–pb | #Opt | Average time (s)* | Max time (s)* | #Opt | Average time (s) | Max time (s) | #Opt |
| 5 | 500 | 44.17 | 218.18 | 10 | 0.43 | 0.72 | 2 | 10 | 0.31 | 0.49 | 10 | 4.11 | 34.45 | 10 |
| | 1000 | 568.37 | 1200.00 | 7 | 0.51 | 0.66 | 1 | 10 | 0.92 | 1.06 | 10 | 192.25 | 1200.00 | 9 |
| | 2500 | 1106.42 | 1200.00 | 1 | 0.98 | 1.28 | 1 | 10 | 5.34 | 5.71 | 10 | 3.25 | 11.40 | 10 |
| | 5000 | 929.04 | 1200.00 | 3 | 1.57 | 1.84 | 1 | 10 | 20.81 | 21.52 | 10 | 126.53 | 1200.00 | 9 |
| | 10000 | 987.01 | 1200.00 | 2 | 3.03 | 3.67 | 1 | 10 | 83.93 | 85.19 | 10 | 17.98 | 58.52 | 10 |
| 10 | 500 | 71.73 | 423.98 | 10 | 0.46 | 0.64 | 2 | 10 | 1.50 | 11.32 | 10 | 1.11 | 5.83 | 10 |
| | 1000 | 1200.00 | 1200.00 | 0 | 0.47 | 0.67 | 2 | 10 | 1.27 | 1.38 | 10 | 0.61 | 0.91 | 10 |
| | 2500 | 1200.00 | 1200.00 | 0 | 0.84 | 1.01 | 1 | 10 | 7.33 | 7.72 | 10 | 121.28 | 1200.00 | 9 |
| | 5000 | 825.85 | 1200.00 | 4 | 1.47 | 1.62 | 1 | 10 | 29.18 | 30.52 | 10 | 633.70 | 1200.00 | 5 |
| | 10000 | 1200.00 | 1200.00 | 0 | 3.10 | 3.48 | 1 | 10 | 149.73 | 154.61 | 10 | 966.57 | 1200.00 | 2 |
| 20 | 500 | 382.23 | 1200.00 | 7 | 0.61 | 1.14 | 2 | 10 | 0.56 | 0.78 | 10 | 6.04 | 55.21 | 10 |
| | 1000 | 50.76 | 229.96 | 10 | 0.51 | 0.87 | 1 | 10 | 2.15 | 2.63 | 10 | 1.60 | 9.33 | 10 |
| | 2500 | 1200.00 | 1200.00 | 0 | 0.88 | 1.40 | 2 | 10 | 13.01 | 13.68 | 10 | 1.18 | 1.69 | 10 |
| | 5000 | 1054.83 | 1200.00 | 2 | 1.58 | 1.95 | 1 | 10 | 53.45 | 54.99 | 10 | 19.53 | 173.16 | 10 |
| | 10000 | 1200.00 | 1200.00 | 0 | 2.96 | 3.74 | 1 | 10 | 346.58 | 353.68 | 10 | 143.82 | 1200.00 | 9 |
| 30 | 500 | 237.75 | 1200.00 | 9 | 1.62 | 4.73 | 5 | 10 | 0.76 | 0.89 | 10 | 5.51 | 37.41 | 10 |
| | 1000 | 499.63 | 1200.00 | 8 | 0.87 | 1.95 | 2 | 10 | 3.32 | 3.63 | 10 | 2.44 | 17.33 | 10 |
| | 2500 | 1175.79 | 1200.00 | 1 | 0.99 | 1.25 | 2 | 10 | 19.58 | 20.20 | 10 | 3.39 | 13.21 | 10 |
| | 5000 | 380.40 | 1200.00 | 8 | 1.62 | 2.59 | 1 | 10 | 79.76 | 83.42 | 10 | 4.73 | 23.68 | 10 |
| | 10000 | 907.74 | 1200.00 | 5 | 4.82 | 8.07 | 2 | 10 | 526.61 | 549.03 | 10 | 37.36 | 249.21 | 10 |

Table 4: KPS benchmark instances (from [7]): time (s) and number of optima.

These instances involve a lower number of families and show up to be harder for CPLEX 12.5 than the previous ones. Nevertheless, even though CPLEX 12.5 runs out of time for most of the large instances, our method is able to find all optima with limited computational effort. The dynamic programming algorithm is capable of reaching all the optima as well. However the computational times are much larger and increase with the size of the instances. We remark that the tests in [7] were carried out on a slightly less performing machine (an asterisk is introduced in the table to point out that times refer to another machine, namely an Intel core TMi3 CPU @ 2.1 GHZ with 2GB of RAM). Anyhow given these results, it is very reasonable to assume that the differences in the performances would remain significant

even if the algorithms were launched on the same machine.

Finally, we tested the scalability of our approach on larger instances with $e_1 = e_2$ uniformly ranging in the interval [0.15, 0.25], $w_{ij}$ integer uniformly distributed in the range [10,100] and $p_{ij} = w_{ij} + 10$, $b = 0.5 \left( \sum_{i=1}^{N} \sum_{j=1}^{n_i} w_{ij} \right)$ with the number of families and items up to 200 and 100000 respectively. The results are reported in Table 5.

| | | CPLEX 12.5 | | | Exact approach | | | |
|---|---|---|---|---|---|---|---|---|
| | | Average | Max | | Average | Max | Max | |
| $N$ | $n$ | time (s) | time (s) | #Opt | time (s) | time (s) | #Sub–pb | #Opt |
| 5 | 20000 | 380.64 | 1200.00 | 7 | 6.93 | 12.20 | 1 | 10 |
| | 50000 | 630.15 | 1200.00 | 5 | 55.49 | 97.38 | 1 | 10 |
| | 100000 | 752.66 | 1200.00 | 6 | 285.74 | 542.94 | 1 | 10 |
| 10 | 20000 | 374.49 | 1200.00 | 7 | 4.38 | 8.21 | 1 | 10 |
| | 50000 | 427.40 | 1200.00 | 7 | 27.14 | 52.92 | 1 | 10 |
| | 100000 | 731.80 | 1200.00 | 5 | 135.99 | 390.62 | 1 | 10 |
| 20 | 20000 | 1031.29 | 1200.00 | 2 | 6.35 | 9.16 | 1 | 10 |
| | 50000 | 1190.44 | 1200.00 | 1 | 26.98 | 43.01 | 1 | 10 |
| | 100000 | 1095.40 | 1200.00 | 1 | 102.92 | 231.63 | 1 | 10 |
| 30 | 20000 | 736.75 | 1200.00 | 4 | 9.40 | 14.49 | 1 | 10 |
| | 50000 | 749.91 | 1200.00 | 4 | 31.84 | 39.87 | 1 | 10 |
| | 100000 | 1092.11 | 1200.00 | 2 | 127.39 | 179.67 | 1 | 10 |
| 50 | 20000 | 685.84 | 1200.00 | 5 | 8.27 | 14.81 | 2 | 10 |
| | 50000 | 1196.78 | 1200.00 | 1 | 51.06 | 87.31 | 2 | 10 |
| | 100000 | 1139.12 | 1200.00 | 1 | 147.89 | 218.74 | 2 | 10 |
| 100 | 20000 | 750.62 | 1200.00 | 5 | 18.34 | 59.61 | 2 | 10 |
| | 50000 | 1116.64 | 1200.00 | 1 | 89.39 | 497.89 | 1 | 10 |
| | 100000 | 1090.48 | 1200.00 | 1 | 128.84 | 272.81 | 2 | 10 |
| 200 | 20000 | 367.69 | 1200.00 | 8 | 19.94 | 43.13 | 2 | 10 |
| | 50000 | 966.37 | 1200.00 | 3 | 105.39 | 284.50 | 1 | 10 |
| | 100000 | 1113.78 | 1200.00 | 1 | 163.02 | 359.18 | 2 | 10 |

Table 5: KPS larger instances: time (s) and number of optima.

We notice that our approach effectively applies also to these larger instances, requiring approximately 540 seconds for the worst-case instance with 100000 items. CPLEX 12.5 fails to reach the optimum within the time limit of 1200 seconds in more than 60% of the instances of Table 5.

These extensive computational experiments confirm the the effectiveness of our exact approach, which strongly outperforms CPLEX 12.5 and the algorithms reported in the literature. This approach is capable of solving to optimality all instances within the time limit.

## 5. Conclusions

In this paper we propose an exact approach for KPS based on an effective exploration of a specific set of variables that leads to solving standard knapsack problems. The presented approach proves to be very effective

and capable of handling instances with up to 100000 items and 200 families with little computational effort while previous approaches were limited to instances with up to 10000 items. The approach outperforms CPLEX 12.5 and favorably compares to the algorithms available in literature.

In future work we will investigate to what extent the proposed approach could be applied to other variants of KPS and to other combinatorial optimization problems involving two sets of variables.

**Acknowledgements**

This work has been partially supported by FLEXMETER, FLEXible smart METERing for multiple energy vectors with active prosumers, funded by the European Commission under H2020, Grant Agreement N. 646568 (see [10]).

[1] J. Adams, E. Balas, D. Zawack. The Shifting Bottleneck Procedure for Job-Shop Scheduling. *Management Science*, 34, 3, 391–401, (1988).

[2] N. Altay, P.E. Robinson Jr., K.M. Bretthauer. Exact and heuristic solution approaches for the mixed integer setup knapsack problem, *European Journal of Operational Research*, 190, 598–609, (2008).

[3] U. Akinc. Approximate and exact algorithm for the fixed–charge knapsack problem, *European Journal of Operational Research*, 170, 363–375, (2004).

[4] J. Carlier. The One-Machine Sequencing Problem. *European Journal of Operational Research*, 11, 42–47, (1982).

[5] M. Caserta, E. Quinonez Rico, A. Marquez Uribe. A cross entropy algorithm for the knapsack problem with setups, *Computers and Operations Research*, 35, 241–252, (2008).

[6] E.D. Chajakis, M. Guignard. Exact algorithms for the setup knapsack problem, *INFOR*, 32, 124–142, (1994).

[7] K. Chebil, M. Khemakhem. A dynamic programming algorithm for the Knapsack Problem with Setup, *Computers and Operations Research*, 64, 40–50, (2015).

[8] F. Della Croce, A. Grosso. Simplex Algorithms for Linear Programming, In: *Concepts of Combinatorial Optimization*, WILEY, 135–155, (2010).

[9] F. Della Croce, F. Salassa. Improved LP-based algorithms for the closest string problem, *Computers and Operations Research*, 39, 746–749, (2012).

[10] http://flexmeter.polito.it.

[11] H. Kellerer, U. Pferschy, D. Pisinger. *Knapsack Problems*, Springer, (2004).

[12] E. Lin. A bibliographical survey on some well-known non-standard knapsack problems. *INFOR*, 36, 274–317, (1998).

[13] S. Martello, D. Pisinger, P. Toth. New trends in exact algorithms for the 0–1 knapsack problems. *European Journal of Operational Research*, 123, 325–332. (2000).

[14] S. Martello, P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*, Wiley, (1990).

[15] L.A. McLay, S. H. Jacobson. Algorithms for the bounded set-up knapsack problem. *Discrete Optimization*, 4, 206–212, (2007).

[16] S. Michel, N. Perrot, F. Vanderbeck. Knapsack problems with setups. *European Journal of Operational Research*, 196, 909–918, (2009).

[17] R.M. Nauss. An Efficient Algorithm for the 0-1 Knapsack Problem. *Management Science*, 23, 27–31, (1976).

[18] E. Horowitz, S. Sahni. Computing partitions with applications to the knapsack problem. *Journal of the ACM* 21, 277–292, (1974).

[19] Y. Yang, R.L. Bulfin. An exact algorithm for the Knapsack Problem with Setup. *Int. J. Operational Research*, 5, 280–291, (2009).