Heuristic and exact reduction procedures to solve the discounted 0–1 knapsack problem

Christophe Wilbaut, Raca Todosijevic, Saïd Hanafi, Arnaud Fréville

Heuristic and exact reduction procedures to solve the discounted 0–1 knapsack problem

Christophe Wilbaut, Raca Todosijevic, Saïd Hanafi, Arnaud Fréville

Discrete Optimization

Heuristic and exact reduction procedures to solve the discounted 0–1 knapsack problem

Christophe Wilbaut[a,b,*], Raca Todosijevic[a,b], Saïd Hanafi[a,b], Arnaud Fréville[a,1]

[a] Univ. Polytechnique Hauts-de-France, LAMIH, CNRS, UMR 8201, Valenciennes F-59313, France
[b] INSA Hauts-de-France, Valenciennes F-59313, France

ARTICLE INFO

Article history:
Received 7 March 2021
Accepted 22 April 2022
Available online xxx

Keywords:
Combinatorial optimization
Discounted knapsack problem
Fixation
Dynamic programming
Instance generator

ABSTRACT

In this paper we consider the discounted 0–1 knapsack problem (DKP), which is an extension of the classical knapsack problem where a set of items is decomposed into groups of three items. At most one item can be chosen from each group and the aim is to maximize the total profit of the selected items while respecting the knapsack capacity constraint. The DKP is a relatively recent problem in the literature. It was considered in several recent works where metaheuristics are implemented to solve instances from the literature. In this paper we propose a two-phase approach in which the problem is reduced by applying exact and / or heuristic fixation rules in a first phase that can be viewed as a preprocessing phase. The remaining problem can then be solved by dynamic programming. Experiments performed on available instances in the literature show that the fixation techniques are very useful to solve these instances. Indeed, the preprocessing phase greatly reduces the size of these instances, leading to a significant reduction in the time required for dynamic programming to provide an optimal solution. Then, we generate a new set of instances that are more difficult to solve by exact methods. The hardness of these instances is confirmed experimentally by considering both the use of CPLEX solver and our approach.

1. Introduction

Knapsack problems arise in many applications from various areas and several variants were derived from the original knapsack problem (KP). The KP is defined as follows. Given a set N of n items, where each item $j \in N$ has associated profit c_j and weight a_j, the aim of the KP is to select a subset of N in order to maximize the profit of the selected items without exceeding a given capacity b of the knapsack. The KP is among the optimization problems with the simplest linear integer programming formulation since it can be formulated with only one capacity (or resource) constraint. The binary (0–1) variant of the KP may be defined by using binary variables x_j, associated to each item j, that specify if item j is selected ($x_j = 1$) or not ($x_j = 0$). Then, the 0–1 KP is given as follows:

$$\text{(0-1 KP)} \begin{cases} \max & \sum_{j=1}^{n} c_j x_j & (1) \\ \text{subject to:} & \sum_{j=1}^{n} a_j x_j \leq b & (2) \\ & x_j \in \{0,1\} \qquad j \in N = \{1,\ldots,n\} & (3) \end{cases}$$

The KP has been intensively studied from the middle of the twentieth century, starting especially with works of Lorie & Savage (1955), Gilmore & Gomory (1966) or Nemhauser & Ullmann (1969). The success of the KP is due to the fact that it can be extended easily to several interesting and challenging variants by adding side constraints (e.g., the multidimensional knapsack problem), by changing the objective (e.g., the quadratic knapsack problem) or by partitioning the set of variables (e.g., knapsack problem with setup) for instance. In fact the KP appears frequently as a subproblem of other hard optimization problems and a wide range of practical applications can be listed for these family of problems such as cargo loading, cutting stock, capital budgeting or project selection for instance. Several review papers and books were dedicated to the KP and some of its variants (see for instance

* Corresponding author.
E-mail addresses: christophe.wilbaut@uphf.fr (C. Wilbaut), raca.todosijevic@uphf.fr (R. Todosijevic), said.hanafi@uphf.fr (S. Hanafi).
[1] *"Professeur émérite"* - Now retired.

https://doi.org/10.1016/j.ejor.2022.04.036

Kellerer, Pferschy, & Pisinger (2004); Salkin & Kluyver (1975); Wilbaut, Hanafi, & Salhi (2007) among others). The discounted 0–1 knapsack problem (DKP) is one of the extensions of the 0–1 KP in which variables are partitioned. It was introduced in 2007 by Guldan in his Master thesis (Guldan, 2007). The word *discounted* comes from the promotional discounts activities of real merchants: in the same way as buying two items at the same time during the promotion season, a discount is defined when buying both. In the DKP items are grouped by three and the value of the discounted item is the sum of the other two items in the group. In addition the discounted item consumes less resources than the total depletion of the other two items. Finally, it is allowed to select at most one item from the group. The aim of the DKP is to maximize the total value associated with the selected items without exceeding the knapsack capacity. Formally, the DKP involves a set M of m groups of three items denoted by $N_i = \{3i, 3i+1, 3i+2\}$ for $i \in M = \{0, \ldots, m-1\}$, which thus contains items to be possibly packed into the knapsack of capacity b. Each item $3i+k$ in group N_i, $k \in \{0,1,2\}$ and $i \in M$, is characterized by a profit c_{3i+k} and a weight a_{3i+k} such that $c_{3i+2} = c_{3i} + c_{3i+1}$ and $\max\{a_{3i}, a_{3i+1}\} < a_{3i+2} < a_{3i} + a_{3i+1}$. The aim of the DKP is to choose at most one item from each group N_i such that the total profit of chosen elements is maximized without having the weight sum to exceed b. In the following we use notation $N = \{0, \ldots, 3m-1\}$ to refer to the index set of items whereas $n = |N| = 3m$ is the number of items. As in the KP all the coefficients c_j, a_j, $\forall j \in N$ and the capacity b are integer and are assumed to be non negative. A linear formulation of the DKP (Rong, Figueira, & Klamroth, 2012) is obtained by introducing three binary variables x_{3i}, x_{3i+1} and x_{3i+2} associated with each group $i \in M$, such that $x_{3i+k} = 1$ if and only if the k^{th} item of group i is selected, for $k \in \{0,1,2\}$. Hence, the standard 0–1 linear programming formulation of DKP is given as follows:

$$
(\text{DKP}) \begin{cases} \max & \displaystyle\sum_{i \in M} c_{3i}x_{3i} + c_{3i+1}x_{3i+1} + c_{3i+2}x_{3i+2} & (4) \\ \text{s.t.:} & \displaystyle\sum_{i \in M} a_{3i}x_{3i} + a_{3i+1}x_{3i+1} + a_{3i+2}x_{3i+2} \leq b & (5) \\ & x_{3i} + x_{3i+1} + x_{3i+2} \leq 1 \qquad i \in M & (6) \\ & x_{3i}, x_{3i+1}, x_{3i+2} \in \{0,1\} \qquad i \in M & (7) \end{cases}
$$

In this model objective function (4) aims to maximize the total profit of selected items. Constraint (5) is the knapsack constraint related to the $n = 3m$ items in the problem, whereas constraints (6) force the choice of at most one item from each group $i \in M$. The all variables are binary variables as stated in (7). As mentioned in Rong et al. (2012) additional conditions can be imposed to avoid trivial solutions and to guarantee that each item in a given group has a chance to be selected. In particular it is generally assumed that $a_{3i} < a_{3i+1} < a_{3i+2}$, $c_{3i} < c_{3i+1} < c_{3i+2}$, and $a_{3i} + a_{3i+1} > a_{3i+2}$. Finally, it is also assumed that $\sum_{i \in M} a_{3i+2} > b$ and $a_{3i+2} \leq b, i \in M$. It may be observed that by deleting constraints (6) the DKP reduces to the KP. As the KP is an NP-hard problem, this implies the DKP is NP-hard as well (Guldan, 2007). The DKP can also be viewed as a particular case of the multiple-choice KP (MCKP) where items are also grouped. However, unlike to the DKP, in the MCKP groups do not necessarily contain the same number of items. In addition specific relations between items in a group are defined in the DKP, which is not the case in the MCKP.

The literature on the DKP started with Guldan (2007) where Guldan presented an exact algorithm based on dynamic programming (Bellman, 1957). Then, Rong et al. (2012) presented a natural formulation of the DKP and solved it by combining the core concept used to solve the KP with the dynamic programming. They proposed three variants based on partitioning the problem according to the type of instance and using or not dominance rules. The results showed that the detection of dominated states leads to an increase of the average running time needed to solve an instance. He et al. also used dynamic programming to solve the DKP in He, Wang, He, Zhao, & Li (2016b). Authors derived a recursive formula based on the principle of minimizing the total weight rather than maximizing the total profit. This new variant of dynamic programming has clearly a lower complexity than the previous one when the sum of profit coefficients is less than the knapsack capacity. They also proposed a fully polynomial-time approximation scheme, a 2-approximation algorithm and a particle swarm optimization (PSO) heuristic for the DKP. The fully polynomial-time approximation scheme is based on the construction of a new instance by modifying only the set of original profit coefficients. This new instance is then solved by a dynamic programming algorithm. Then, authors proposed a greedy algorithm and show that it is a 2-approximation algorithm for the DKP. They demonstrated that the relationship between the ratios of the three items in a given group can be induced by only four cases and they exploited this property in the greedy algorithm. Finally, authors proposed a greedy repair algorithm to deal with infeasible solutions and incorporate it in a PSO algorithm. Some other references related to evolutionary algorithms dedicated to the DKP can be found in the literature. He et al. proposed in He, Wang, Li, Zhang, & Chen (2016a) two elitist genetic algorithms and two types of greedy strategies to repair and to optimize the individuals. Feng et al. proposed in Feng, Wang, Li, & Li (2018) a multi-strategy monarch butterfly optimization heuristic for solving the DKP. They introduced two effective strategies based on a neighborhood mutation with crowding and a Gaussian perturbation to improve the behavior of the approach. Then, Feng and Wang designed ten types of moth search in Feng & Wang (2018). Zhu et al. considered in Zhu, He, Wang, & Tsang (2017) three differential evolution algorithms for the DKP that mainly differ in the encoding mechanisms used to represent (feasible) solutions of the problem. The first variant is an adaptation of the hybrid-encoding binary differential evolution algorithm proposed in He, Wang, & Kou (2007) to solve the KP and the SAT problem. Authors also introduced the repair operator proposed in He et al. (2016a) to manage infeasible solutions. This variant mimics the classical formulation of the DKP with $3m$ binary variables by using an encoding conversion function to transform a $3m$-dimensional real vector into a $3m$-dimensional binary vector. The other two variants exploit another encoding in which a feasible solution is an integer vector in $\{0,1,2,3\}^m$ allowing a direct application of the standard differential evolution mechanism. Authors considered two different encoding conversion functions corresponding to the two variants. More recently, He et al. proposed in He, Wang, & Gao (2019) a new kind of method to design an evolutionary algorithm by using algebraic theory. The proposition is mainly based on two evolution operators using the addition, multiplication and inverse operation of the direct product of rings. The first one is called global exploration operator and build a new individual by learning from four different elements that are randomly selected from the whole search space. The second one is called local development operator and is based on random changes in a given individual. In the new evolutionary algorithm authors propose to apply these two operators successively and to replace an existing individual in the population if the new one produced by these operations has a better fitness. The approach is adapted to the DKP by using the repair operator proposed in He et al. (2016a). He and Wang considered very recently a similar approach to solve the set-union knapsack problem, the bounded knapsack problem and the DKP (He & Wang, 2021) and obtained competitive results compared with He et al. (2016a). Wu et al. proposed in Wu, Zhao, Feng, & Lee (2020) a discrete hybrid teaching-learning-based optimization algorithm. They introduced self-learning factors in the teacher and learner phases, which balances the exploitation and exploration of the algorithm. They also used to types of crossover to improve

the search. The proposed method is evaluated and compared with 7 other population-based algorithms ((Feng et al., 2018; Feng & Wang, 2018; He et al., 2016a; He et al., 2016b; Wu, He, & Chen, 2017; Zhu et al., 2017)) on a set of 80 available instances in the literature. The results showed that their approach obtained in average the best solutions. We also noted the recent works in Truong (2021a,b) where the author considered some specific components in PSO and a new moth-flame optimization algorithm for the DKP. The proposed approaches obtained better results in average than some of the works in the literature.

In this paper we consider reduction techniques to solve the DKP efficiently. One of the objective of this work is to show that such techniques can be used in a heuristic or an exact process to obtain near optimal or optimal solutions of the input problem in a very short time. More precisely, the main contributions of this work are: (*i*) we propose heuristic and exact fixation rules for the DKP that can be used in a preprocessing phase in order to reduce the size of a given instance; (*ii*) we show that using these techniques is an efficient way to solve all the existing instances in the literature in just a few seconds when applying dynamic programming; (*iii*) we propose a new variant of a generator to provide harder instances. We demonstrate that these instances are effectively harder to solve when using a solver like CPLEX. The new instances are available for the community so that colleagues can consider them in future works.

The paper is organized as follows. In Section 2 we present the fixation rules that can be applied in a preprocessing phase to reduce the size of the initial problem. Then, we provide in Section 3 computational results over two sets of instances from the literature when the reduced problem is solved with dynamic programming. Section 4 is devoted to the use of the new generator to provide harder instances and the experimental evaluation performed both with CPLEX and our approach. Section 5 concludes the paper.

2. Fixation rules

Fixation techniques and reduction rules are often used to solve optimization problems in general and knapsack problems in particular. In many cases the aim is to set values for a subset of variables before solving the reduced problem with a dedicated exact method (see e.g., Rong & Figueira (2013)). In other cases a decomposition method is applied for solving more efficiently the initial problem (Chen & Hao, 2014; Dahmani, Hifi, & Wu, 2016). Another solution is to solve the reduced problem obtained when fixing a subset of variables with a metaheuristic (Wilbaut, Hanafi, Fréville, & Balev, 2006). Considering the KP several approaches were proposed to reduce the size before applying a branch-and-bound method or a dynamic programming algorithm (see, e.g., Martello & Toth (1988); Pisinger (1997)), whereas other implicit enumeration algorithms include a reduction phase (see, e.g., Fayard & Plateau (1975, 1982)). In this section we consider the use of both heuristic and exact fixation rules to solve the DKP. In particular, we consider techniques originally proposed for solving the MCKP or other knapsack variants to the DKP case. We start this section with the notion of LP-dominance that can be used to set variables' values in a heuristic way. Then we present a simple reduction process to remove definitively some groups from the initial problem without eliminating any optimal solution of the problem.

2.1. LP-Dominance

The first fixation rule is adapted from an existing result for the MCKP. It is based on the notion of *LP-dominance* which can be used to solve efficiently the LP-relaxation of the MCKP (Sinha & Zoltners, 1979). As mentioned previously the DKP can be viewed as a special case of the MCKP. Then, a given instance P of DKP can be formulated as a special instance P' of MCKP by introducing for each group a dummy item with a null profit and a null weight. More formally, we construct P' from the original DKP instance by considering m groups of four items $N'_i = \{4i, 4i+1, 4i+2, 4i+3\}$ for $i \in M$ such that $c'_{i,k} = c_{3i+k-1}$ and $a'_{i,k} = a_{3i+k-1}$ for $k \in \{1,2,3\}$ and $c'_{i,0} = a'_{i,0} = 0 \ \forall i \in M$. Then, by introducing a binary variable $x_{i,k}$ which takes value 1 if and only if item k in group N'_i (i.e., item $4i+k$) is chosen, for $k \in \{0,1,2,3\}$, the problem instance P' can be formulated as:

$$
(P') \begin{cases}
\max & \displaystyle\sum_{i\in M}\sum_{k=0}^{3} c'_{i,k}x_{i,k} & & (8)\\
\text{s.t.:} & \displaystyle\sum_{i\in M}\sum_{k=0}^{3} a'_{i,k}x_{i,k} \le b & & (9)\\
& \displaystyle\sum_{k=0}^{3} x_{i,k} = 1 & i \in M & (10)\\
& x_{i,k} \in \{0,1\} & i \in M, k \in \{0,1,2,3\} & (11)
\end{cases}
$$

Problem instance P' can be viewed as a reformulation of the previous standard model DKP where the m groups are explicitly mentioned in the objective function and in the knapsack constraint, leading to a double sum in (8) and in (9). The particularity of this model comes from the use of the dummy items allowing us to force the choice of exactly one item from each group (10). Please note that in the rest of the paper we use both notations $4i+k$ or (i,k) to refer to item k in group N'_i.

When dealing with the MCKP the LP-dominance may be used to set some variables at value 0 in an optimal solution of the LP-relaxation. One of the interest of this property is that it can be used as a pre-processing phase since it is only based on the data (Sinha & Zoltners, 1979). The following definition introduces the LP-dominance when considering instance P' from the original definition for the MCKP.

Definition 1. Let P' be the reformulated instance of the DKP and let $i \in M$ be a given group of items. If some items $j,k,l \in N'_i$ with $a'_{i,j} < a'_{i,k} < a'_{i,l}$ and $c'_{i,j} < c'_{i,k} < c'_{i,l}$ satisfy

$$\frac{c'_{i,l} - c'_{i,k}}{a'_{i,l} - a'_{i,k}} \ge \frac{c'_{i,k} - c'_{i,j}}{a'_{i,k} - a'_{i,j}} \tag{12}$$

then item k is said to be LP-dominated by items j and l.

From this initial definition we can observe that when dealing with the DKP we know by hypothesis that the profits and the weights of items are always in non-decreasing order into each group, from the fictive item to the last one (i.e., $c'_{i,0} < c'_{i,1} < c'_{i,2} < c'_{i,3}$ and $a'_{i,0} < a'_{i,1} < a'_{i,2} < a'_{i,3}$ for every $i \in M$). In addition only items $(i,1)$ and $(i,2)$ can be dominated in a group. Those observations lead to the following restricted definition.

Definition 2. Let P' be the reformulated instance of the DKP and let $i \in M$ be a given group of items. Item (i,1) is LP-dominated if and only if (13) or (14) is satisfied.

$$\frac{c'_{i,2} - c'_{i,1}}{a'_{i,2} - a'_{i,1}} \ge \frac{c'_{i,1}}{a'_{i,1}} \tag{13}$$

$$\frac{c'_{i,3} - c'_{i,1}}{a'_{i,3} - a'_{i,1}} \ge \frac{c'_{i,1}}{a'_{i,1}} \tag{14}$$

In the same way item (i,2) is LP-dominated if and only if (15) or (16) is satisfied.

$$\frac{c'_{i,3} - c'_{i,2}}{a'_{i,3} - a'_{i,2}} \ge \frac{c'_{i,2} - c'_{i,1}}{a'_{i,2} - a'_{i,1}} \tag{15}$$

$$\frac{c'_{i,3} - c'_{i,2}}{a'_{i,3} - a'_{i,2}} \geq \frac{c'_{i,2}}{a'_{i,2}} \tag{16}$$

From this definition we can apply the following proposition (a proof can be found for instance in Sinha & Zoltners (1979)).

Proposition 1. *Let $i \in M$ be a group of items. If item (i,1) (resp. (i,2)) is LP-dominated then an optimal solution to the LP relaxation of P' with $x_{i,1} = 0$ (resp. $x_{i,2} = 0$) exists.*

The characteristics of the DKP allow us not to have to sort the items into the groups, leading to a very short pre-processing phase. Please note that another dominance rule exists to set some variables definitively at their optimal value in an optimal solution of the MCKP (see Sinha & Zoltners (1979)). It can be applied if there exist items (i, j) and (i, j') in a group i satisfying $c'_{i,j} \geq c'_{i,j'}$ and $a'_{i,j} \leq a'_{i,j'}$. However, in the case of the DKP the corresponding propriety cannot be satisfied (here again due to the hypothesis on the profits and the weights). In this paper we apply Proposition 1 to set some variables at value 0 in a heuristic way. To achieve this we adapt the algorithm proposed by Zemel (1980) to solve the LP-relaxation of the MCKP. This approach first reformulates the original LP-relaxation of the MCKP into a corresponding LP-relaxation of a KP. Then, this relaxation can be solved efficiently based on the well-known greedy algorithm. Proposition 1 can be applied while solving the LP-relaxation. Our method is described in Algorithm 1 . It consists in the same two main steps. The first step aims at building the KP instance composed by at most $3m$ items (since we can eliminate some variables according to Proposition 1). In the KP instance we associate with each item $k \in \{1, 2, 3\}$ in group $i \in M$ of the DKP a profit noted $c''_{i,k}$ and a weight $a''_{i,k}$. The procedure to build the KP was initially presented and justified by Zemel in Zemel (1980). In the case of the DKP it can be summarized as follows. Based on the fact that items are already sorted in each group according to their weight in the knapsack constraint, the fictive item is first eliminated in all the groups. Then, when considering a group composed only by items that are not LP-dominated, (Eqs. 17–18) and (19–20) are applied respectively to compute the profit and weight values in the KP instance:

$$c''_{i,1} = c'_{i,1} \tag{17}$$

$$c''_{i,k} = c'_{i,k} - c'_{i,k-1} \qquad k = 2, 3 \tag{18}$$

$$a''_{i,1} = a'_{i,1} \tag{19}$$

$$a''_{i,k} = a'_{i,k} - a'_{i,k-1} \qquad k = 2, 3 \tag{20}$$

The incremental profit $c''_{i,k}$ in group k is a measure of how much we gain if item k is chosen instead of item $k-1$. The incremental weight $a''_{i,k}$ has a similar interpretation. One can observe that if a given item is LP-dominated then the procedure is adapted to discard this value as shown in Algorithm 1. In this algorithm we use the following convention: an LP-dominated item has its profit and weight fixed to 0 in the KP instance. The step consisting of eliminating the LP-dominated items and building the KP instance is described between line 2 and line 10. The LP-dominated items are added into a set denoted to as F^0 (and defined in line 1) which thus contains variables that can be set at 0 in an optimal solution of the LP-relaxation. In the second step of the algorithm (from line 11 to line 24) the LP-relaxation of the KP instance is solved with a greedy algorithm when items are ordered according to their incremental efficiencies denoted to as $e''_{i,k}$ in Algorithm 1. By convention the efficiency of an LP-dominated item is set to $-\infty$ so that it would not be added in the solution (see lines 4 and 7). During this solving a feasible solution is also obtained by discarding the fractional item in the LP solution. Line 19 is used to avoid the selection of more than one item in a given group, whereas lines 20 to 24 manage the case where the LP solution has two fractional items.

Algorithm 1: Solving the LP-relaxation and building a greedy solution.

Function `LP-Greedy ()`

1 $F^0 \leftarrow \oslash$;

`/* Step 1: Eliminate LP-dominated items and build an instance of KP */`

2 **for** *i from 1 to m* **do**

`/* Check if item (i, 1) is LP-dominated */`

3 **if** *equation (13) or equation (14) is satisfied* **then**

4 $c''_{i,1} = a''_{i,1} \leftarrow 0$; $e''_{i,1} \leftarrow -\infty$; $F^0 \leftarrow F^0 \cup (i, 1)$;

else

5 $c''_{i,1} \leftarrow c'_{i,1}$; $a''_{i,1} \leftarrow a'_{i,1}$; $e''_{i,1} \leftarrow c''_{i,1}/a''_{i,1}$;

end

`/* Check if item (i, 2) is LP-dominated */`

6 **if** *equation (15) or equation (16) is satisfied* **then**

7 $c''_{i,2} = a''_{i,2} \leftarrow 0$; $e''_{i,2} \leftarrow -\infty$; $F^0 \leftarrow F^0 \cup (i, 2)$;

8 $c''_{i,3} \leftarrow c'_{i,3} - c''_{i,1}$; $a''_{i,3} \leftarrow a'_{i,3} - a''_{i,1}$; $e''_{i,3} \leftarrow c''_{i,3}/a''_{i,3}$;

else

9 $c''_{i,2} \leftarrow c'_{i,2} - c''_{i,1}$; $a''_{i,2} \leftarrow a'_{i,2} - a''_{i,1}$; $e''_{i,2} \leftarrow c''_{i,2}/a''_{i,2}$;

10 $c''_{i,3} \leftarrow c'_{i,3} - c''_{i,2}$; $a''_{i,3} \leftarrow a'_{i,3} - a''_{i,2}$; $e''_{i,3} \leftarrow c''_{i,3}/a''_{i,3}$;

end

end

`/* Step 2: Solve the KP instance in a greedy way */`

11 $\bar{v} = \underline{v} \leftarrow 0$; $\bar{b} \leftarrow b$; $j \leftarrow 1$;

12 $\bar{x}_{i,k} = x_{i,k} \leftarrow 0$, $\forall i \in M, k \in \{1, 2, 3\}$;

13 Sort the items according to non-increasing order of $e''_{i,k}$ values, $\forall i \in M, k \in \{1, 2, 3\}$;

14 **while** $\bar{b} > 0$ **do**

15 Let (i', k') the original indexes of next item in the order ;

16 **if** $\bar{b} > a''_{i',k'}$ **then**

17 $\bar{v} \leftarrow \bar{v} + c''_{i',k'}$; $\bar{b} \leftarrow \bar{b} - a''_{i',k'}$;

18 $\bar{x}_{i',k'} = x_{i',k'} \leftarrow 1$;

19 $\bar{x}_{i',k} = x_{i',k} \leftarrow 0, \forall k \in \{1, 2, 3\}$ such that $k \neq k'$;

else

20 $\underline{v} \leftarrow \bar{v}$;

21 $\bar{x}_{i',k'} \leftarrow \bar{b}/a''_{i',k'}$; $\bar{v} \leftarrow \bar{v} + c''_{i',k'}\bar{x}_{i',k'}$; $\bar{b} \leftarrow 0$;

22 **if** $\exists k \neq k'$ *in group* i' *such that* $\bar{x}_{i',k} = 1$ **then**

23 $\bar{x}_{i',k} \leftarrow 1 - \bar{x}_{i',k'}$;$x_{i',k} \leftarrow 0$;

end

end

24 $j \leftarrow j + 1$;

end

`/* Step 3: Try to fill the feasible solution */`

25 **while** $j \leq 3m - |F^0|$ **do**

26 Let (i', k') the original indexes of next item in the order ;

27 **if** $\bar{b} > a''_{i',k'}$ *and* $x_{i',k} = 0, \forall k \in \{1, 2, 3\}$ **then**

28 $x_{i',k'} \leftarrow 1$;$\underline{v} \leftarrow \underline{v} + c''_{i',k'}$; $\bar{b} \leftarrow \bar{b} - a''_{i',k'}$;

end

29 $j \leftarrow j + 1$;

end

30 Return $(\bar{x}, \bar{v})$, $(x, \underline{v})$ and F^0;

In that case they are necessarily in the same group (see Proposition 3 in Zemel (1980)). The last part of the algorithm (from line 25 to line 29) can be used to try adding a few items in the current feasible solution x in a greedy way, where we only consider non LP-dominated items and groups where no item were previously selected. Variables returned by Algorithm 1 in set F^0 can be set definitively in the problem if we want to use the fixation as a heuristic.

This algorithm can be used to solve the LP-relaxation of the problem and to build an initial solution in a very short time. In addition, set F^0 can be used to fix variables in a heuristic way, and thus to reduce the initial problem, as we will see in the computational results. In the next section we introduce another property that can be used to reduce the size of the problem by fixing definitively variables at their optimal value.

2.2. Reducing the number of groups

In the previous section we presented a technique that can be used to fix variables in a heuristic way. In this section we consider another technique to set variables definitively at their optimal value in an optimal solution of the DKP. Thus in that case the fixation is valid for the original problem. This fixation rule comes from a well-known property often used when solving variants of knapsack problems. Let P be an instance of the DKP and let $(P|x_{i,k} = \alpha)$ be instance P in which we fix only variable k in group i at $\alpha \in \{0, 1\}$, for $i \in M$ and $k \in \{1, 2, 3\}$. In addition, let $\overline{v}(P|x_{i,k} = \alpha)$ be an upper bound of problem $(P|x_{i,k} = \alpha)$ and $\underline{v}$ be a lower bound on the optimal value of the original problem P. Finally, let y be the feasible solution associated with $\underline{v}$. We can then apply the following Property 1, where notation $\lfloor \gamma \rfloor$ is used to refer to the rounding function, which returns the maximum integer number not greater than γ.

Property 1. *If $\lfloor \overline{v}(P|x_{i,k} = \alpha) \rfloor \leq \underline{v}$, then there exists an optimal solution x^* of problem P where $x^*_{i,k} = 1 - \alpha$ or solution y is optimal for problem P.*

Our idea is to exploit the optimal solution $\bar{x}$ of the LP-relaxation of the DKP provided by Algorithm 1 as a starting point. In fact, we do not consider all the variables in the fixation process. Indeed, when considering the DKP the most interesting result is the fixation of one variable in a group since that allows to remove this group since the other two variables are automatically set to value 0. In addition, according to the special structure of the DKP and the coefficients in the problem it is clear that in a given group i variable $x_{i,3}$ is the one for which the fixation is most probable. Thus, we apply Property 1 only with $\alpha = 0$ and for every item $(i, 3)$, $i \in M$ such that $\bar{x}_{i,3} = 1$. Indeed, if $\bar{x}_{i,3} = 0$ for a given i then the value $\lfloor \overline{v}(P|x_{i,3} = \alpha) \rfloor = \lfloor \bar{v} \rfloor$. Even if in practice most of items $(i, 3)$ are chosen in solution $\bar{x}$ in the worst case we thus need to solve m LP-relaxations.

Solving the LP-relaxation of problem $(P|x_{i,3} = 0)$ for a given group i requires some adjustments in steps 1 and 2 of Algorithm 1. Indeed, some LP-dominated items in the original problem are not yet dominated when item $(i, 3)$ is eliminated (when variable $x_{i,3}$ is set to 0), thus implying some modifications in the construction of the knapsack instance and in the order of items when considering the efficiency measure. However, it is not necessary to restart from scratch and step 1 can be almost avoided since only items $(i, 1)$ and $(i, 2)$ can be impacted. Once the KP instance is built its LP-relaxation can be solved with the same principle as in Algorithm 1 to provide both the upper bound $\overline{v}(P|x_{i,3} = 0)$ and a lower bound for problem $(P|x_{i,3} = 0)$ which is also valid for the initial DKP. We use notations $\overline{v}_i$ (resp. $\underline{v}_i$) for this upper (resp. lower) bound and notation xg_i for the greedy solution corresponding to this lower bound. Thus, during the process we generate several greedy solutions, in such a way we may improve our initial feasible solution and improve the number of variables that can be set. The procedure is summarized in Algorithm 2.

Algorithm 2: Reducing the number of groups in the DKP.

Function `UB-Fix`$(\bar{x}, \underline{v}, F^0)$
1 $F^1 \leftarrow \emptyset$; $\underline{v}^{best} \leftarrow \underline{v}$ **for** *i from 1 to m* **do**
3 **if** $\bar{x}_{i,3} = 1$ **then**
   `/* Adapt the construction of the KP instance only for group i */`
4 Save the values of $c''_{i,k}, a''_{i,k}$ and $e''_{i,k}$ in the KP instance associated with $\bar{x}$, $\forall k \in \{1, 2, 3\}$;
5 **if** $(i, 1) \in F^0$ **then**
6 **if** *equation 13 is satisfied* **then**
7 $c''_{i,1} = a''_{i,1} \leftarrow 0$; $e''_{i,1} \leftarrow -\infty$;
   else
8 $c''_{i,1} \leftarrow c'_{i,1}$; $a''_{i,1} \leftarrow a'_{i,1}$; $e''_{i,1} \leftarrow c''_{i,1}/a''_{i,1}$;
   end
   else
9 $c''_{i,1} \leftarrow c'_{i,1}$; $a''_{i,1} \leftarrow a'_{i,1}$; $e''_{i,1} \leftarrow c''_{i,1}/a''_{i,1}$;
10 $c''_{i,2} \leftarrow c'_{i,2} - c''_{i,1}$; $a''_{i,2} \leftarrow a'_{i,2} - a''_{i,1}$; $e''_{i,2} \leftarrow c''_{i,2}/a''_{i,2}$;
   end
11 $c''_{i,3} \leftarrow 0$; $a''_{i,3} \leftarrow 0$; $e''_{i,3} \leftarrow -\infty$;
   `/* Solve the LP and obtain the upper and lower bounds */`
12 Apply Steps 2 and 3 of Algorithm 1 to solve the current KP instance ;
13 Obtain $\overline{v}_i$, $\underline{v}_i$ and xg_i ;
14 **if** $\underline{v}_i \geq \underline{v}^{best}$ **then**
15 $\underline{v}^{best} \leftarrow \underline{v}_i$; $x^{best} \leftarrow xg_i$;
   end
16 **if** $\lfloor \overline{v}_i \rfloor \leq \underline{v}^{best}$ **then**
17 $F^1 \leftarrow F^1 \cup \{i\}$;
   end
18 Restore values $c''_{i,k}, a''_{i,k}$ and $e''_{i,k}$, $\forall k \in \{1, 2, 3\}$;
  end
19 Return F^1, $\underline{v}^{best}$, x^{best} ;
end

In this algorithm we consider as inputs the results of Algorithm 1, in particular the optimal solution $\bar{x}$ of the LP-relaxation of the DKP, the initial lower bound $\underline{v}$, the set F^0 to know LP-dominated items. To simplify the presentation we also suppose that the KP instance (i.e., vectors c'', a'' and e'') built to solve the LP-relaxation remains available. We use the following notations. Set F^1 refers to the set of groups that can be eliminated by fixing the corresponding last item at value 1. Thus, we just store in this set the index i of such a group if necessary. Notation $\underline{v}^{best}$ (resp. x^{best}) is used to refer to the best lower bound (resp. feasible solution) during the process.

As explained above the main loop in the algorithm explores the m sub-problems restricted to those where $\bar{x}_{i,3} = 1$. Between lines 4 and 11 we first modify the KP instance by checking if items that were previously LP-dominated are always dominated. In fact in our case when considering group i at the current iteration item $(i, 2)$ cannot be LP-dominated since it is the last item in the group. In addition item $(i, 1)$ can be only dominated by item $(i, 2)$, thus only Eq. 13 can be satisfied.

Then, we apply steps 2 and 3 of Algorithm 1 to obtain the current upper (resp. lower) bound $\overline{u}_i$ (resp. $\underline{v}_i$) and the corresponding feasible solution (lines 12 and 13). Once we have the bounds we

then can check if a better lower bound (and feasible solution) has been obtained and if the current group i can be definitively fixed according to Property 1 between lines 14 and 17. The algorithm returns the set F^1, the best lower bound and the corresponding best feasible solution.

Fixation rules described in this section can be used in different ways. For example it is possible to fix LP-dominated variables according to Proposition 1 and then to solve the reduced problem. In that case we say that the fixation is heuristic since it is valid when solving the LP-relaxation of the DKP only. Another strategy consists in applying Algorithm 2 and Property 1 to fix definitively some variables x_{3i+2} and thus to eliminate the corresponding groups. In that case we say the fixation is optimal since this property is valid when solving the original DKP.

In our experiments we consider the use of these two techniques with dynamic programming (DP) to solve the instances from the literature. Dynamic programming was already applied on the DKP in Rong et al. (2012) and in He et al. (2016b). Rong et al. presented in Rong et al. (2012) a natural extension of the sequential DP algorithm from the classical 0–1 KP to the DKP. In that case the principle is to look for maximizing the total profit value with the given sum of weight coefficients. Later, He et al. proposed in He et al. (2016b) a variant in which the principle is to minimize the total weight with the given sum of value coefficients. The principle of both methods is quite similar and their complexity depends mainly on the size of the problem (in particular the value b or the value $\sum_{i\in M} c_{3i+2}$). In this paper we use the natural recursion formula based on the capacity of the knapsack b. For the DKP the DP process is based on m stages where each stage corresponds to a set of three variables, where at most one variable can be set to value one. This property allows the use of a similar structure of the DP algorithm to that for solving the knapsack problem. Then, a stage is made up of $b+1$ states defined by the recursive equation according the data of the problem. For the sake of clarity we reintroduce this recursion formula (Rong et al., 2012). Let us denote by $v_i(\beta)$ the objective value of state $\beta \in \{0,\dots,b\}$ at stage $i \in \{0,\dots,m-1\}$. When $i=0$ we have :

$$v_0(\beta) = \begin{cases} 0 & \text{if } 0 \le \beta < a_0 \\ c_0 & \text{if } a_0 \le \beta < a_1 \\ c_1 & \text{if } a_1 \le \beta < a_2 \\ c_2 & \text{if } a_2 \le \beta \le b \end{cases} \tag{21}$$

Then, values $v_i(\beta)$ associated with the following stages $1,\dots,m-1$ and $\beta = 0,\dots,b$ can be defined as follows:

$$v_i(\beta) = \begin{cases} v_{i-1}(\beta) & \text{if } 0 \le \beta < a_{3i} \\ \max\{v_{i-1}(\beta), v_{i-1}(\beta - a_{3i}) + c_{3i}\} & \text{if } a_{3i} \le \beta < a_{3i+1} \\ \max\{v_{i-1}(\beta), v_{i-1}(\beta - a_{3i}) & \\ \quad +c_{3i}, v_{i-1}(\beta - a_{3i+1}) + c_{3i+1}\} & \text{if } a_{3i+1} \le \beta < a_{3i+2} \\ \max\{v_{i-1}(\beta), v_{i-1}(\beta - a_{3i}) & \\ \quad +c_{3i}, v_{i-1}(\beta - a_{3i+1}) + c_{3i+1}, & \\ \quad v_{i-1}(t - a_{3i+2}) + c_{3i+2}\} & \text{if } a_{3i+2} \le \beta \le b \end{cases} \tag{22}$$

The optimal value corresponds to $v_{m-1}(b)$.

It is important to note that in the rest of the paper when we use the DP to solve a given instance we use a classical implementation of the previous recursive equations and we do not implement any optimization or dominance techniques. For instance we use the traditional way to find an optimal solution by applying a backtracking phase through the set of states. This technique needs $m \times (b+1)$ memory space to save a variable value (i.e., an integer to know which item is selected from the group associated with the stage) for each state corresponding to each stage. Our objective is to show that combining the techniques based on the fixation of variables with DP is an efficient approach to solve the DKP.

3. Solving the instances from the literature

In this section we consider the instances available in the literature to evaluate our approaches. All the algorithms presented in this paper were coded in C++ language, implemented and compiled with `Visual Studio` tools on a Windows 10 platform. The tests were carried out on an HP `EliteBook` with 8GB of RAM and an `Intel CORE i7` processor with 2.60GHz. We used `-O2 -Oi -Ot` options in `Visual Studio` to optimize the code and to further accelerate the program. All the CPU times reported were obtained using `clock` function and `CLOCKS_PER_SEC` macro. We used `CPLEX 12.7.1.0` with the `Concert` technology with default parameters and a limited running time of 1 h, unless otherwise stated, to solve the instances.

Rong et al. (2012) provided some characteristics and dominance rules between items according to the correlation of the data and the difficulty of the instance. Then they proposed a generator for three types of instances: uncorrelated, weakly correlated and strongly correlated, respectively. The results obtained by Rong et al. (2012) showed that: i) correlated instances are easier to solve than uncorrelated instances for the DKP and ii) strongly correlated instances are easier to solve than weakly correlated instances. It can be observed that the instances used in most of the papers available in the literature are those provided later by He et al. (2016b). He et al. proposed a few adjustments to the generator of Rong et al. and developed a fourth type of instances, so-called inverse strongly correlated instances. All available instances are organized into two data sets (or groups), each one containing 40 instances: 10 instances for each correlation type with m varying between 100 and 1000.

It should be noted that papers in the literature dealing with metaheuristics to solve the DKP used these data sets to evaluate the performance of the proposed approaches. We provide in Table 1 a synthesis of the results observed with these two data sets. Results related to data set 1 are in the left part of Table 1 whereas data set 2 is considered in the right part. We first report in this table the time required by CPLEX to solve each instance. Then, column DP corresponds to the time needed to solve the instance with the initial dynamic programming algorithm (i.e., without any reduction). Next two columns report the percentage of variables that are LP-dominated according to Proposition 1 (column `%Dom.`) and the percentage of groups that can be eliminated by fixing variable x_{3i+2} at value 1 according to Property 1 (column `%Fix.`), respectively. Finally, column DP+ gives the time needed to solve the instance if we consider dynamic programming with the reduction rule (i.e., by deleting groups via Algorithm 2).

Values reported in Table 1 clearly show that all the available instances in the literature can be solved instantly by a software like CPLEX. If the use of such an optimization tool is not possible the results also show that dynamic programming is an interesting alternative when considering these instances. We can observe a real difference with values reported in columns DP: the DP algorithm is able to solve all the instances in data set 2 in less than 3 seconds, whereas the running time increases until approximately one minute for a few instances in data set 1. That can be explained by the generator used to produce the instances. As mentioned previously He et al. proposed in He et al. (2016b) a variant of the generator proposed in Rong et al. (2012) to evaluate the advantage of both recurrence formula. As a result the value of b in the first set of instances is largely superior, inevitably leading to an increased running time when using the natural recursion formula (interesting readers are invited to consult (He et al., 2016b) for more details). In practice high values for coefficient b can be a limit for the DP algorithm according to the resources available on the machine test. It was for example impossible for us to solve instance

Table 1
Computational results over data sets 1 and 2 from the literature.

data set 1						data set 2					
Instance	CPLEX	DP	%Dom.	%Fix.	DP+	Instance	CPLEX	DP	%Dom.	%Fix.	DP+
udkp1_1	0.10	0.22	49.33	68.00	0.05	udkp2_1	0.04	0.06	46.00	60.00	0.02
udkp1_2	0.12	0.68	47.67	49.50	0.13	udkp2_2	0.07	0.13	44.50	47.50	0.04
udkp1_3	0.24	1.93	46.33	61.00	0.21	udkp2_3	0.31	0.30	44.56	68.00	0.04
udkp1_4	0.28	4.11	46.50	64.25	0.46	udkp2_4	0.20	0.52	46.25	61.00	0.05
udkp1_5	0.24	8.60	46.27	54.00	0.69	udkp2_5	0.33	0.88	43.73	62.00	0.15
udkp1_6	0.25	12.49	46.56	50.33	0.83	udkp2_6	0.37	1.41	44.94	64.83	0.20
udkp1_7	0.24	27.39	46.19	58.57	1.07	udkp2_7	0.58	1.70	47.14	67.29	0.24
udkp1_8	0.29	36.61	46.96	51.25	1.38	udkp2_8	1.17	1.85	46.00	58.25	0.30
udkp1_9	0.26	58.61	46.22	60.78	1.79	udkp2_9	0.49	2.28	45.74	58.56	0.30
udkp1_10	0.37	O/M[a]	46.93	69.50	1.38	udkp2_10	0.34	2.76	45.27	50.80	0.55
wdkp1_1	0.08	0.16	65.33	60.00	0.02	wdkp2_1	0.11	0.06	58.00	57.00	0.01
wdkp1_2	0.09	0.40	64.83	45.50	0.07	wdkp2_2	0.12	0.17	57.83	55.00	0.02
wdkp1_3	0.07	1.10	63.56	60.00	0.05	wdkp2_3	0.16	0.32	57.33	66.00	0.04
wdkp1_4	0.12	1.84	64.58	52.75	0.12	wdkp2_4	0.22	0.45	58.67	61.50	0.06
wdkp1_5	0.14	6.36	64.20	69.60	0.11	wdkp2_5	0.20	0.93	58.73	65.20	0.10
wdkp1_6	0.12	9.22	65.00	68.00	0.15	wdkp2_6	0.11	1.11	58.89	63.33	0.10
wdkp1_7	0.28	8.85	65.10	50.71	0.34	wdkp2_7	0.32	1.48	58.33	63.71	0.18
wdkp1_8	0.18	20.52	64.63	68.88	0.38	wdkp2_8	0.52	1.74	59.08	58.88	0.20
wdkp1_9	0.17	22.89	64.41	60.78	0.42	wdkp2_9	0.56	2.50	58.56	56.89	0.23
wdkp1_10	0.26	31.01	64.53	55.80	0.57	wdkp2_10	0.54	2.52	58.77	54.50	0.26
sdkp1_1	0.05	0.20	60.67	65.00	0.02	sdkp2_1	0.20	0.08	54.00	66.00	0.01
sdkp1_2	0.04	0.52	60.67	52.50	0.06	sdkp2_2	0.10	0.21	53.50	61.50	0.02
sdkp1_3	0.08	1.29	60.00	58.33	0.16	sdkp2_3	0.27	0.24	52.89	58.00	0.05
sdkp1_4	0.36	2.30	60.17	67.25	0.09	sdkp2_4	0.17	0.48	53.42	62.00	0.06
sdkp1_5	0.19	4.19	60.60	70.80	0.18	sdkp2_5	0.24	0.79	52.53	65.40	0.16
sdkp1_6	0.17	4.07	60.83	62.50	0.24	sdkp2_6	0.17	1.02	50.89	57.00	0.15
sdkp1_7	0.32	12.44	60.52	66.86	0.28	sdkp2_7	0.23	1.82	52.24	65.14	0.23
sdkp1_8	0.24	13.94	60.13	58.38	0.41	sdkp2_8	0.27	1.74	53.08	61.13	0.27
sdkp1_9	0.41	29.38	60.22	70.00	0.39	sdkp2_9	0.23	2.17	52.63	60.89	0.31
sdkp1_10	0.21	31.01	60.47	58.10	0.56	sdkp2_10	0.30	2.24	51.83	59.00	0.36
idkp1_1	0.05	0.16	63.33	51.00	0.03	idkp2_1	0.07	0.06	61.00	51.00	0.01
idkp1_2	0.07	0.54	64.50	52.50	0.04	idkp2_2	0.10	0.22	58.50	48.50	0.01
idkp1_3	0.20	1.48	65.22	65.67	0.09	idkp2_3	0.18	0.39	59.78	62.33	0.03
idkp1_4	0.16	2.26	64.75	59.50	0.16	idkp2_4	0.14	0.68	60.25	58.50	0.04
idkp1_5	0.20	3.20	65.47	51.80	0.26	idkp2_5	0.14	0.92	59.53	57.20	0.04
idkp1_6	0.15	6.27	65.06	67.67	0.24	idkp2_6	0.20	1.29	60.06	64.00	0.07
idkp1_7	0.22	17.18	64.95	66.14	0.22	idkp2_7	0.21	1.40	59.90	54.86	0.13
idkp1_8	0.32	26.17	65.00	71.25	0.30	idkp2_8	0.23	1.78	59.71	48.63	0.25
idkp1_9	0.20	24.90	64.74	50.44	0.47	idkp2_9	0.27	2.09	59.89	47.56	0.15
idkp1_10	0.44	41.63	64.80	63.30	0.65	idkp2_10	0.27	2.76	59.67	46.40	0.23

[a] Out of Memory.

udkp1_10 of set 1 as mentioned in Table 1 by O/M for "Out of Memory".

Values reported in columns %Dom. indicate that a large part of the variables are LP-dominated. That can be used for example to solve the problem after fixing the corresponding variables in a heuristic way. In that case we can solve the reduced problem with DP to build very quickly a feasible solution. According to the results obtained both by CPLEX and the basic DP algorithm we do not report the corresponding values. We observed that applying this heuristic fixation before the DP algorithm leads to the visit of 75 optimal solutions over the 80 instances. Thus, this technique can be considered as a good heuristic if the size of the instance remains reasonable.

An interesting result is related to columns DP+. Indeed the values in these columns clearly indicate that applying the reduction procedure described in Algorithm 2 followed by the DP algorithm is a good strategy to solve these instances. Even if the running times is in general superior to the one observed with CPLEX we can notice that it is always less than 2 seconds. To the best of our knowledge these results overpass previously published results for solving these two sets of instances. It should be emphasized that DP+ solved all instances from the literature unlike standard DP.

The results presented in this section show that available instances of DKP in the literature do not offer a high level of opposition for exact approaches. Some additional experiments showed us that CPLEX can fix in general more than 90% of the variables. Concerning our approach values reported in column %Fix. show that in general more than half of the groups can be deleted thanks to Algorithm 2. Thus, we concluded that these instances are probably too impacted by reduction mechanisms. In the next section we propose a variant of the existing generator to provide harder instances.

4. Generating and solving harder instances

In this section we first propose to adapt the generators proposed in He et al. (2016b); Rong et al. (2012) to produce harder DKP instances. The generator is described in Section 4.1. Then, we consider both the use of CPLEX and our dynamic programming-based approaches to solve these instances. The corresponding results are reported in Section 4.2.

4.1. A new generator

As mentioned previously two papers deal with a generator for DKP instances (He et al., 2016b; Rong et al., 2012). Rong et al. proposed in Rong et al. (2012) a first generator, whereas He et al. proposed in He et al. (2016b) some modifications. Both generators were used to build the two sets of instances in the literature. In this paper we propose to adapt the generator of He et al. to build

more complicated instances. To achieve this we tighten the range of values of the weights of item. This should make it possible to limit the fixing of variables x_{3i+2} at value one, and thus to prevent having a reduced problem that is too small to solve. For example in the case of uncorrelated instances authors in He et al. (2016b) proposed to select randomly coefficients c_{3i} and c_{3i+1} in group i [128,3072]. As $c_{3i+2} = c_{3i} + c_{3i+1}$ in DKP this wide range of values is highly likely to favor the third object of the group. This is reinforced by the selection of the weights a_{3i} and a_{3i+1} in [256,4098] (with $a_{3i} < a_{3i+1}$) and $a_{3i+2} \in [a_{3i+1} + 1, a_{3i} + a_{3i+1} + 1]$. With these ranges of values item $3i+2$ in group i can have a weight slightly greater than that of item $3i+1$ for a much greater coefficient in the objective function. Preliminary experiments showed that the difference between coefficients of the three items has to be small enough to balance the selection chance among the group. These tests confirmed that difficult instances can be obtained using [4000,4200] to select a_{3i} and a_{3i+1}, and using [3000,3200] to choose c_{3i} and c_{3i+1}. According to these observations and to remain with values of the same order of magnitude as those mentioned in He et al. (2016b) we decided to use these intervals for all types of instances. As shown above the definition of weight a_{3i+2} is also important in order to complicate the instance. In this work we consider a more general rule by selecting this weight in interval $[a_{3i+1} + (1-\epsilon)a_{3i} + 1, a_{3i} + a_{3i+1} + 1[$, where ϵ is a decimal number in [0,1]. When $\epsilon = 1$ we obtain the rule proposed in He et al. (2016b) with a possible large interval. On the contrary if $\epsilon = 0$ then $a_{3i+2} = a_{3i} + a_{3i+1}$ corresponding to an extreme case. Experiments confirmed that, in general, the closer the value ϵ is to 0, the more difficult the instances are. Thus, we decided to set $\epsilon = 0.01$ to generate the instances without enforcing $a_{3i+2} = a_{3i} + a_{3i+1}$. Finally, we use the same rules as in He et al. (2016b) to set coefficients c_{3i} and c_{3i+1} in correlated instances. The capacity of the knapsack is based on the same formula as in He et al. (2016b) with parameter $r \in [0.25, 0.75]$ rather than [0.5,0.75] to increase the range of possible values. As in the previous works we consider four kinds of instances: uncorrelated instances, weakly correlated instances, strongly correlated instances, and inverse strongly correlated instances. For each type of instances, the profit coefficients, weight coefficients, and knapsack capacity are generated as follows:

1. uncorrelated instances:
   - $a_{3i} \in [4000, 4200[$, $a_{3i+1} \in [4000, 4200]$ with $a_{3i} < a_{3i+1}$, $a_{3i+2} \in [a_{3i+1} + (1-\epsilon)a_{3i} + 1, a_{3i} + a_{3i+1} + 1[$, with ϵ a decimal number in [0,1].
   - $c_{3i} \in [3000, 3200[$, $c_{3i+1} \in [3000, 3200]$ with $c_{3i} < c_{3i+1}$, and $c_{3i+2} = c_{3i} + c_{3i+1}$
   - $b = r \sum_{i=0}^{m-1} w_{3i+2}$, with r a random decimal number in [0.25, 0.75].
2. weakly correlated instances:
   - $a_{3i} \in [4000, 4200[$, $a_{3i+1} \in [4000, 4200]$ with $a_{3i} < a_{3i+1}$, $a_{3i+2} \in [a_{3i+1} + (1-\epsilon)a_{3i} + 1, a_{3i} + a_{3i+1} + 1[$, with ϵ a decimal number in [0,1].
   - $c_{3i} \in [a_{3i} - 100, a_{3i} + 100[$, $c_{3i+1} \in [a_{3i+1} - 100, a_{3i+1} + 100]$ with $c_{3i} < c_{3i+1}$, and $c_{3i+2} = c_{3i} + c_{3i+1}$
   - $b = r \sum_{i=0}^{m-1} w_{3i+2}$, with r a random decimal number in [0.25, 0.75].
3. strongly correlated instances:
   - $a_{3i} \in [4000, 4200[$, $a_{3i+1} \in [4000, 4200]$ with $a_{3i} < a_{3i+1}$, $a_{3i+2} \in [a_{3i+1} + (1-\epsilon)a_{3i} + 1, a_{3i} + a_{3i+1} + 1[$, with ϵ a decimal number in [0,1].
   - $c_{3i} = a_{3i} + 100$, $c_{3i+1} = a_{3i+1} + 100$, and $c_{3i+2} = c_{3i} + c_{3i+1}$
   - $b = r \sum_{i=0}^{m-1} w_{3i+2}$, with r a random decimal number in [0.25, 0.75].
4. inverse strongly correlated instances:
   - $c_{3i} \in [4000, 4200[$, $c_{3i+1} \in [4000, 4200]$ with $c_{3i} < c_{3i+1}$, and $c_{3i+2} = c_{3i} + a_{3i+1}$.
   - $a_{3i} = c_{3i} + 100$, $a_{3i+1} = c_{3i+1} + 100$, and $a_{3i+2} \in [a_{3i+1} + (1-\epsilon)a_{3i} + 1, a_{3i} + a_{3i+1} + 1[$
   - $b = r \sum_{i=0}^{m-1} w_{3i+2}$, with r a random decimal number in [0.25, 0.75], with ϵ a decimal number in [0,1].

We vary the number of groups m from 100 to 1 000, and we generate 10 instances for every value of m, leading to 100 instances for each type, and 400 instances in total. The new data set is available on the Internet[2]. The size of these instances are similar to those existing in the literature. This makes it easy to compare the behavior of algorithms between the sets of instances.

4.2. Solving the new instances with CPLEX or dynamic programming

We report in Tables 2 and 3 a synthesis of the results obtained over the 400 new instances. In these tables a row contains the results over 10 distinct instances related to every value of m. In columns CPLEX we report the number of instances the solver was able to solve in a maximum running time of 1 h (column `solved`) and the number of solutions returned that are not optimal (column `not opt.`), respectively[3]. Then, column DP gives the number of instances that can be solved with the basic DP algorithm (i.e., without using fixation) without encountering a memory problem. Column `%Fix.` have the same meaning as in Table 1, whereas values in column DP+ report the number of instances that can be solved when applying Algorithm 2 before DP.

Tables 2 and 3 seem to indicate that uncorrelated instances and weakly correlated instances are, in general, harder to solve than strongly correlated and inverse strongly correlated ones for the DKP. Indeed, the number of instances not solved by CPLEX and the number of non optimal solutions provided by the solver are more important for these two correlation types. This observation was made in other works when dealing with instances from the literature (He et al., 2016b; Rong et al., 2012). An interesting observation from values in these two columns is the fact that it is possible to generate some relatively hard instances of DKP even for small values of n, when considering exact methods based on branch-and-bound. However, when the number of groups is small DP can always be applied without any problem since the profit, weight and knapsack values are obviously reasonable. Despite this we can also observe that the basic DP cannot solve all the instances with $n = 800$ in our experiments.

As shown by values in column `%Fix.` having tightened the weights of the objects clearly reduces the fixation of variables x_{3i+2}, which explains the difficulty of these instances compared to those of the literature. This lower fixation can be observed for all the correlation type, even if the difference with values in Table 1 is less important in general for the inverse strongly correlated instances. The fact less groups can be dropped from the instances impacts logically the capability of the DP+ method to solve all the instances. However values reported in this column are interesting since they show that the reduction phase can be used to help the DP algorithm to solve more instances for all type of correlation. More precisely, DP algorithm can solve instances with $m \leq 700$ in less than one minute, in general. When the reduction rule can be applied then a few larger instances can be solved as shown in columns DP+. In that case the algorithm requires at most 30 seconds. Thus, the combination of the reduction rule and the use of dynamic programming makes it possible to solve more instances than CPLEX for the largest uncorrelated and weakly correlated

[2] https://oae.uphf.fr/content/UVHC/paAEJHcO0.

[3] CPLEX may find an optimal solution without being able to prove its optimality in the allotted time

Table 2
Solving new uncorrelated and weakly correlated instances with CPLEX and DP.

uncorrelated instances						weakly correlated instances					
m	CPLEX		DP	%Fix.	DP+	m	CPLEX		DP	%Fix.	DP+
	solved	not opt.					solved	not opt.			
100	0	3	10	0.00	10	100	5	0	10	0.00	10
200	3	4	10	4.60	10	200	0	5	10	0.00	10
300	4	1	10	5.40	10	300	0	5	10	0.00	10
400	1	1	10	0.43	10	400	2	2	10	8.20	10
500	3	4	10	10.00	10	500	3	0	10	9.18	10
600	4	3	10	14.62	10	600	5	1	10	0.12	10
700	3	4	10	5.09	10	700	3	2	10	14.16	10
800	5	3	7	14.04	8	800	4	1	9	14.83	10
900	4	0	8	8.98	9	900	2	0	1	3.87	2
1000	4	3	1	17.99	5	1000	2	1	3	0.00	3

Table 3
Solving new strongly correlated and inverse strongly correlated instances with CPLEX and DP.

strongly correlated instances						inverse strongly correlated instances					
m	CPLEX		DP	%Fix.	DP+	m	CPLEX		DP	%Fix.	DP+
	solved	not opt.					solved	not opt.			
100	10	0	10	0.90	10	100	10	0	10	16.70	10
200	1	5	10	0.00	10	200	10	0	10	23.70	10
300	5	2	10	6.57	10	300	9	1	10	22.43	10
400	8	0	10	3.98	10	400	9	1	10	19.75	10
500	8	0	10	0.00	10	500	9	0	10	20.96	10
600	8	1	10	5.93	10	600	10	0	10	21.83	10
700	4	3	10	0.00	10	700	10	0	10	4.87	10
800	8	0	7	18.71	8	800	9	0	4	19.14	4
900	10	0	3	24.37	5	900	10	0	1	5.54	2
1000	5	1	1	9.60	3	1000	10	0	1	14.31	3

Table 4
Results over new uncorrelated and weakly correlated instances with heuristics.

uncorrelated instances						weakly correlated instances					
m	CPLEX	DP as a heuristic			Algo.2	m	CPLEX	DP as a heuristic			Algo.2
	avg. gap	%Dom.	nb.opt.	avg. gap	avg. gap		avg. gap	%Dom.	nb.opt.	avg. gap	avg. gap
100	0.269	40.77	1	0.234	0.617	100	0.028	43.17	6	0.028	0.673
200	0.031	39.50	5	0.018	0.200	200	0.032	43.82	8	0.016	0.295
300	0.028	39.24	4	0.019	0.154	300	0.019	43.74	5	0.013	0.172
400	0.009	41.64	6	0.008	0.102	400	0.006	39.13	6	0.006	0.098
500	0.007	36.93	7	0.004	0.065	500	0.002	38.88	9	0.002	0.051
600	0.003	35.28	9	0.002	0.039	600	0.002	43.51	8	0.002	0.083
700	0.003	39.28	8	0.003	0.056	700	0.001	36.33	8	0.001	0.039
800	0.002	35.52	6	N/A	0.042	800	0.001	35.92	8	N/A	0.041
900	0.002	37.67	7	N/A	0.038	900	0.001	41.28	1	N/A	0.035
1000	0.001	33.83	2	N/A	0.029	1000	0.001	43.40	3	N/A	0.063

instances we considered in these data sets. The situation is not the same for strongly correlated instances where CPLEX can solve more larger instances (but less instances with $m \leq 700$) and for inverse strongly correlated instances. For this last category CPLEX is not able to find only 2 optimal solutions over the 100 instances. Globally the difference between the time required by the basic DP and the DP+ variant is very important. In fact detailed results showed that the reduction rule, when it can be applied, helps in reducing the running time of DP by 70% to 83% on average according to the correlation type of the instances, leading to a significant improvement.

Since some instances cannot be solved by CPLEX and the DP algorithm it can be interesting to consider heuristic approaches to deal with these data sets. We provide in Tables 4 and 5 a synthesis of the results obtained when: (*i*) using DP as a heuristic (i.e., when we apply Proposition 1 to fix LP-dominated variables); (*ii*) applying Algorithm 2 alone. In these tables columns avg. gap refer to the average percentage gap of solutions returned by a given algorithm from the optimal LP-solutions. On a given instance this gap is computed as $100 \times \frac{LP-lb}{LP}$, where LP is the optimal value of the LP-relaxation of the problem and lb is the lower bound corresponding to the value of the solution returned by the algorithm. Thus, columns CPLEX provide the average gap of solutions provided by CPLEX. Then, the next three columns give information when using DP as a heuristic: column %Dom. corresponds to the percentage of LP-dominated variables that can be fixed at value 0 in a heuristic way, nb. opt. reports the number of solutions returned by DP on the corresponding reduced problem that are op-

Table 5
Results over new strongly correlated and inverse strongly correlated instances with heuristics.

strongly correlated instances						inverse strongly correlated instances					
m	CPLEX	DP as a heuristic			Algo.2	m	CPLEX	DP as a heuristic			Algo.2
	avg. gap	%Dom.	nb.opt.	avg. gap	avg. gap		avg. gap	%Dom.	nb.opt.	avg. gap	avg. gap
100	0.010	65.53	10	0.010	0.233	100	0.002	54.83	10	0.002	0.023
200	0.006	66.10	5	0.005	0.260	200	< 0.001	50.05	10	< 0.001	0.072
300	0.002	61.58	5	0.002	0.176	300	< 0.001	51.09	10	< 0.001	0.053
400	< 0.001	63.31	6	< 0.001	0.093	400	< 0.001	52.81	10	< 0.001	0.037
500	0.001	65.89	9	0.001	0.052	500	< 0.001	52.01	10	< 0.001	0.013
600	< 0.001	62.03	7	< 0.001	0.044	600	< 0.001	51.43	10	< 0.001	0.018
700	0.001	65.87	9	0.001	0.075	700	< 0.001	62.75	10	< 0.001	0.024
800	< 0.001	53.49	4	N/A	0.029	800	< 0.001	53.27	4	N/A	0.019
900	< 0.001	49.76	3	N/A	0.015	900	< 0.001	62.30	1	N/A	0.039
1000	< 0.001	59.61	0	N/A	0.033	1000	< 0.001	56.50	1	N/A	0.033

timal, whereas avg. gap gives the associated average gap. Please note that mention N/A in this column means that we do not compute the average gap since at least one instance cannot be processed with DP (for memory limitation). Finally, column Algo.2 provides the gap of the solution returned by Algorithm 2.

Values in columns %Dom. show that the average percentages of LP-dominated variables in the new instances tend to be lower than those observed with instances from the literature for uncorrelated and weakly correlated instances (see Table 1). This is not always the case for strongly and inverse strongly correlated instances. Values reported in the other two columns related to the use of DP as a heuristic show that this approach can provide better solutions than CPLEX in many cases, in particular for uncorrelated and weakly correlated instances when $m \leq 700$. Thus this approach can be interesting to provide good feasible solutions, even if its application remains limited by the size of the problem. When it is not possible to apply this method we can always use the solution returned by Algorithm 2. The quality of this solution is logically inferior but the corresponding running time is always less than 1 second for all the instances addressed in this section. This solution could be used as a starting solution for another heuristic or metaheuristic for example.

Results presented in this section show that some new instances with $m \leq 1000$ can be difficult to solve, even if a part of them can be sufficiently reduced to be solved by DP. We believe that these 400 new instances made available to the community represent an important data set to consider when developing and validating exact, heuristic and metaheuristic approaches for the DKP.

5. Conclusion

This paper deals with the discounted 0–1 knapsack problem (DKP), an extension of the knapsack problem, where items are grouped by three, and at most one item from a group can be included in a solution. To tackle the problem we proposed two fixation techniques. The first one can be used in a heuristic way by discarding some variables of the problem according to a LP-dominance rule. This process can discard an optimal solution of the problem but it can be used to provide a near optimal feasible solution. The second one removes some groups from the problem, leading to a smaller problem to solve, without discarding any optimal solution of the original problem. Computational experiments were conducted to show that both fixation techniques can be used in a preprocessing phase before applying dynamic programming to solve the instances proposed in the literature in at most two seconds. According to these results we proposed a new generator to provide a set of 400 harder instances that can be considered by heuristics and metaheuristics. The hardness of these instances is demonstrated experimentally by using CPLEX since it was not able to solve an important number of these new instances with a limited running time of one hour. Thus, we hope that these instances can be useful to the community when proposing and evaluating metaheuristics for solving the DKP.

In future works we will consider the use of more sophisticated heuristics and metaheuristics to solve efficiently the new set of instances. The structure of the DKP can also be studied in more details to look for some properties that can help its solving in general.

Acknowledgment

Authors wish to thank the editor and the referees for their valuable and constructive comments which have significantly improved the presentation of this work.

References

Bellman, R. (1957). *Dynamic programming*. Princeton, NJ, USA: Princeton University Press.
Chen, Y., & Hao, J. K. (2014). A “reduce and solve” approach for the multiple-choice multidimensional knapsack problem. *European Journal of Operational Research, 239*, 313–322.
Dahmani, I., Hifi, M., & Wu, L. (2016). An exact decomposition algorithm for the generalized knapsack sharing problem. *European Journal of Operational Research, 252*, 761–774.
Fayard, D., & Plateau, G. (1975). Resolution of the 0–1 knapsack problem: Comparison of methods. *Mathematical Programming, 8*, 272–307.
Fayard, D., & Plateau, G. (1982). Algorithm 47: An efficient algorithm for the solution of the 0–1 knapsack problem. *Computing, 28*, 269–287.
Feng, Y., Wang, G.-G., Li, W., & Li, N. (2018). Multi-strategy monarch butterfly optimization algorithm for discounted {0-1} knapsack problem. *Neural Computing and Applications, 30*(10), 3019–3036.
Feng, Y.-H., & Wang, G.-G. (2018). Binary moth search algorithm for discounted {0-1} knapsack problem. *IEEE Access, 6*, 10708–10719.
Gilmore, P. C., & Gomory, R. E. (1966). The theory and computation of Knapsack functions. *Operations Research, 14*(6), 1045–1074.
Guldan, B. (2007). Heuristic and Exact Algorithms for Discounted Knapsack Problems. Master thesis University of Erlangen-NürnbergGermany.
He, Y., & Wang, X. (2021). Group theory-based optimization algorithm for solving knapsack problems. *Knowledge-Based Systems, 219*, 104445.
He, Y., Wang, X., & Gao, S. (2019). Ring theory-based evolutionary algorithm and its application to d{0-1} KP. *Applied Soft Computing, 77*, 714–722.
He, Y., Wang, X., & Kou, Y. (2007). A binary differential evolution algorithm with hybrid encoding. *Journal of Computer Research and Development, 44*(9), 1476–1484.
He, Y., Wang, X., Li, W., Zhang, X., & Chen, Y. (2016a). Research on genetic algorithm for discounted {0-1} knapsack problem. *Chinese Journal of Computers, 39*(12), 2614–2630.
He, Y.-C., Wang, X.-Z., He, Y.-L., Zhao, S.-L., & Li, W.-B. (2016b). Exact and approximate algorithms for discounted {0-1} knapsack problem. *Information Sciences, 369*, 634–647.
Kellerer, H., Pferschy, U., & Pisinger, D. (2004). *Knapsack problems*. Berlin: Springer Berlin Heidelberg.
Lorie, J. H., & Savage, L. J. (1955). Three problems in rationing capital. *The Journal of Business, 28*(4), 229–239.
Martello, S., & Toth, P. (1988). A new algorithm for the 0–1 knapsack problem. *Management Science, 34*, 633–644.
Nemhauser, G. L., & Ullmann, Z. (1969). Discrete dynamic programming and capital allocation. *Management Science, 15*(9), 494–505.

Pisinger, D. (1997). A minimal algorithm for the knapsack problem. *Operations Research, 45*, 758–767.

Rong, A., & Figueira, J. R. (2013). A reduction dynamic programming algorithm for the bi-objective integer knapsack problem. *European Journal of Operational Research, 231*, 299–313.

Rong, A., Figueira, J. R., & Klamroth, K. (2012). Dynamic programming based algorithms for the discounted {0–1} knapsack problem. *Applied Mathematics and Computation, 218*(12), 6921–6933.

Salkin, H. M., & Kluyver, C. A. D. (1975). The knapsack problem: a survey. *Naval Research Logistics Quarterly, 22*(1), 127–144.

Sinha, A., & Zoltners, A. A. (1979). The multiple choice knapsack problem. *Operations Research, 27*, 503–515.

Truong, T. K. (2021a). Different transfer functions for binary particle swarm optimization with a new encoding scheme for discounted {0-1} knapsack problem. *Mathematical Problems in Engineering, 2021*. Article ID 2864607

Truong, T. K. (2021b). A new moth-flame optimization algorithm for discounted {0-1} knapsack problem. *Mathematical Problems in Engineering, 2021*. Article ID 5092480

Wilbaut, C., Hanafi, S., Fréville, A., & Balev, S. (2006). Tabu search: Global intensification using dynamic programming. *Control and Cybernetics, 35*, 579–598.

Wilbaut, C., Hanafi, S., & Salhi, S. (2007). A survey of effective heuristics and their application to a variety of knapsack problems. *IMA Journal of Management Mathematics, 19*(3), 227–244.

Wu, C., He, Y., & Chen, Y. (2017). Mutated bat algorithm for solving discounted {0-1} knapsack problem. *Journal of Computer Applications (China), 37*(5), 1292–1299.

Wu, C., Zhao, J., Feng, Y., & Lee, M. (2020). Solving discounted {0-1} knapsack problems by a discrete hybrid teaching-learning-based optimization algorithm. *Applied Intelligence, 50*, 1872–1888.

Zemel, E. (1980). The linear multiple choice knapsack problem. *Operations Research, 28*, 1412–1423.

Zhu, H., He, Y., Wang, X., & Tsang, E. C. C. (2017). Discrete differential evolutions for the discounted {0–1} knapsack problem. *International Journal of Bio-Inspired Computation, 10*(4), 219–238.