


## On the Product Knapsack Problem

Claudia D'Ambrosio<sup>1</sup> · Fabio Furini<sup>2</sup>  ·  
Michele Monaci<sup>3</sup> · Emiliano Traversi<sup>4</sup>

Received: 22 September 2016 / Accepted: 21 December 2017 / Published online: 3 January 2018  
© Springer-Verlag GmbH Germany, part of Springer Nature 2018

**Abstract** Given a set of items, each characterized by a profit and a weight, we study the problem of maximizing the product of the profits of the selected items, while respecting a given capacity. To the best of our knowledge this is the first manuscript that studies this variant of the knapsack problem which we call Product Knapsack Problem (PKP). We show that PKP is weakly NP-hard. We propose and implement a Dynamic Programming algorithm and different Mixed Integer Linear and Nonlinear Programming formulations for the PKP. Finally, we present an extensive computational study on a large set of benchmark instances derived from the literature.

**Keywords** Knapsack problem · Dynamic programming · Complexity · Mixed integer (non)linear programming

---

✉ Fabio Furini  
fabio.furini@dauphine.fr

Claudia D'Ambrosio  
dambrosio@lix.polytechnique.fr

Michele Monaci  
michele.monaci@unibo.it

Emiliano Traversi  
emiliano.traversi@univ-paris13.fr

<sup>1</sup> LIX CNRS (UMR7161), École Polytechnique, Palaiseau, France

<sup>2</sup> PSL, Université Paris Dauphine, LAMSADE, Paris, France

<sup>3</sup> DEI, University of Bologna, Bologna, Italy

<sup>4</sup> LIPN, Paris 13, Villetaneuse, France

## 1 Introduction

In many selection problems, one is given a set  $N$  of *items* and is asked to determine a subset of items  $S \subseteq N$  that maximizes a profit function  $f(S)$  under some operational constraints. The most common problem in this family is the well-known *Knapsack Problem* (KP), in which each item  $j$  has associated a weight  $w_j$  ( $\forall j \in N$ ) and the capacity constraint imposes that the total weight of the selected items does not exceed a given positive threshold  $C$ . In addition, each item  $j$  has associated a profit  $p_j$  ( $\forall j \in N$ ) and the objective function asks to maximize the sum of the profits of the selected items, i.e.,  $f(S) = \sum_{j \in S} p_j$ .

The KP can be modeled by the following Integer Linear Program (ILP) formulation:

$$\max \left\{ \sum_{j \in N} p_j x_j : \sum_{j \in N} w_j x_j \leq C, x_j \in \{0, 1\}, j \in N \right\},$$

where each variable  $x_j$  takes value 1 if and only if item  $j$  is inserted in the knapsack. The KP is NP-hard, although in practice fairly large instances can be solved to optimality within low running time. We note that one can assume, without loss of generality, that profits and weights of all items are positive. The reader is referred to [14, 16] for comprehensive surveys on applications and variants of this problem.

Different profit functions  $f(S)$  have been studied in the literature but, surprisingly, the natural variant which considers the product of the item profits has not received much attention yet. Considering generic profits, i.e., the case in which each  $p_j$  can take either a positive or a negative value, the profit function studied in this paper is the following:

$$f(S) = \prod_{j \in S} p_j. \quad (1)$$

More precisely, (1) describes a profit function  $f(S)$  that assigns to each feasible set of items  $S$  the product of the profits of the items in  $S$ . We can now formally define the problem considered in this manuscript, i.e., the *Product Knapsack problem* (PKP), as follows:

(PKP) *input*: A set  $N = \{1, \dots, n\}$  of items, the  $j$ -th item having a weight  $w_j$

and a profit  $p_j$ , and a single knapsack with positive capacity  $C$ .

*objective*: Determine a subset of items  $S \subseteq N$  such that  $\sum_{j \in S} w_j \leq C$  and

$\prod_{j \in S} p_j$  is maximized.

As usual in knapsack problems, we assume that the weight of each item  $j \in N$  is not larger than the capacity  $C$ ; indeed, any item  $k$  with  $w_k > C$  can be removed from consideration as it cannot be inserted in any feasible solution. In the following we will denote by  $N^+ = \{j \in N : p_j > 0\}$  and  $N^- = \{j \in N : p_j < 0\}$  the sets of items with positive and negative profits, respectively. Without loss of generality, we can assume that all profits are different from zero; indeed, including in the solution

an item with null profit would yield a null profit, that we can obtain also by taking no items. Moreover, we note that, without loss of generality, also items with profit in  $(0, 1]$  can be excluded. Indeed, the only situation in which an optimal solution contains an item, say  $k$ , with  $p_k \in (0, 1]$  is the case in which all items have profit in  $(0, 1]$  and  $k$  is the item with largest profit in  $N$  (the only item in the solution).

To the best of our knowledge this is the first paper that specifically addresses the Product Knapsack Problem (PKP); for this reason, we first study the computational complexity of the problem, and then propose effective exact methods to solve it.

*Preprocessing of items with negative weights.* In the following, we will further assume that all items have a non-negative weight. We now show that this assumption can be done without loss of generality. In particular, we will show that for any instance, say  $I = (n, p, w, C)$ , in which some items have negative weights, we can define an equivalent instance, say  $\tilde{I} = (\tilde{n}, \tilde{p}, \tilde{w}, \tilde{C})$ , with only non-negative weights, whose optimal solution allows to recover the optimal solution of the former instance. Our transformation is based on the following trivial observation:

**Observation 1** *An optimal solution to PKP must include an even number of items  $j \in N^-$ .*

Let us assume that items are sorted by non-decreasing weights, i.e., all items with negative weight (if any) precede items with non-negative weight. Let  $\tilde{n}$  and  $\hat{n}$  denote the number of items with negative weight and the number of items that have both negative weight and profit, respectively. For simplicity, we assume that  $\hat{n}$  is even; later, we will show how to modify the transformation in case  $\hat{n}$  is odd.

Given an instance  $I$  with  $\tilde{n} > 0$ , define a new instance  $\tilde{I}$  as follows:  $\tilde{n} = n$ ,  $\tilde{C} = C - \sum_{j \leq \tilde{n}} w_j$ , and

$$\tilde{w}_j = \begin{cases} -w_j & \text{if } j \leq \tilde{n} \\ w_j & \text{otherwise,} \end{cases} \quad \tilde{p}_j = \begin{cases} \frac{1}{p_j} & \text{if } j \leq \tilde{n} \\ p_j & \text{otherwise.} \end{cases}$$

The transformation above allows us to state the following result:

**Proposition 1** *Let  $\tilde{S}^*$  be an optimal solution for instance  $\tilde{I}$ . Then, an optimal solution  $S^*$  for instance  $I$  can be computed as follows:*

$$S^* = \{j \in \tilde{S}^* : j > \tilde{n}\} \cup \{j \notin \tilde{S}^* : j \leq \tilde{n}\}.$$

*Proof* First, we show that  $S^*$  is a feasible solution. Indeed, we have

$$\begin{aligned} \sum_{j \in S^*} w_j &= \sum_{\substack{j \in \tilde{S}^* \\ j > \tilde{n}}} w_j + \sum_{\substack{j \notin \tilde{S}^* \\ j \leq \tilde{n}}} w_j = \sum_{\substack{j \in \tilde{S}^* \\ j > \tilde{n}}} \tilde{w}_j + \sum_{\substack{j \notin \tilde{S}^* \\ j \leq \tilde{n}}} w_j - \sum_{\substack{j \in \tilde{S}^* \\ j \leq \tilde{n}}} w_j + \sum_{\substack{j \in \tilde{S}^* \\ j \leq \tilde{n}}} w_j = \sum_{j \in \tilde{S}^*} \tilde{w}_j \\ &+ \sum_{j \leq \tilde{n}} w_j \leq C \end{aligned}$$

where the last inequality derives from feasibility of solution  $\tilde{S}^*$  with respect to capacity  $\tilde{C}$ .

Let us denote by  $z$  and  $\tilde{z}$  the optimal solution values of instances  $I$  and  $\tilde{I}$ , respectively. We have

$$\begin{aligned} z &= \prod_{j \in S^*} p_j = \prod_{\substack{j \in \tilde{S}^* \\ j > \tilde{n}}} p_j \prod_{\substack{j \notin \tilde{S}^* \\ j \leq \tilde{n}}} p_j = \prod_{\substack{j \in \tilde{S}^* \\ j > \tilde{n}}} p_j \prod_{\substack{j \in \tilde{S}^* \\ j \leq \tilde{n}}} p_j \prod_{\substack{j \in \tilde{S}^* \\ j \leq \tilde{n}}} p_j \prod_{\substack{j \in \tilde{S}^* \\ j \leq \tilde{n}}} \frac{1}{p_j} \\ &= \prod_{\substack{j \in \tilde{S}^* \\ j > \tilde{n}}} \tilde{p}_j \prod_{j \leq \tilde{n}} p_j \prod_{\substack{j \in \tilde{S}^* \\ j \leq \tilde{n}}} \tilde{p}_j = \prod_{j \in \tilde{S}^*} \tilde{p}_j \prod_{j \leq \tilde{n}} p_j = \left( \prod_{j \leq \tilde{n}} p_j \right) \tilde{z}. \end{aligned}$$

Since  $\hat{n}$  is even,  $\tilde{z}$  is multiplied by a positive constant and, accordingly, the values of the two solutions are proportional to each other. This implies that an optimal solution of the first problem corresponds to an optimal solution of the second one, and vice-versa.

We conclude the proof showing the transformation in case  $\hat{n}$  is odd. In this case, we define an additional item with index  $\tilde{n} = n + 1$ , profit  $\tilde{p}_{\tilde{n}} = -M$ , and weight  $\tilde{w}_{\tilde{n}} = 0$ , where  $M$  is a large (positive) number. It is easy to see that, in this case, any optimal solution includes both item  $\tilde{n}$  and an odd number of items from set  $N^-$ . As to the solution values, we have

$$\prod_{j \in S^*} p_j = -\frac{1}{M} \prod_{j \in \tilde{S}^*} \tilde{p}_j \prod_{j \leq \tilde{n}} p_j = \left( -\frac{1}{M} \prod_{j \leq \tilde{n}} p_j \right) \tilde{z}$$

which concludes the proof. □

Thus, we may assume, without loss of generality, that all weights are positive. In addition, we will assume that all input data are integer numbers; fractions, if any, can be handled by multiplying through a proper factor.

*Other nonlinear variants of the Knapsack Problem.* Different variants of the nonlinear knapsack problem have been considered in the literature. For example, [9] proposed heuristics for the general mixed integer nonlinear knapsack problem when the objective function is separable. Another relevant variant is the *Quadratic Knapsack Problem* (QKP). Given an item set  $S$ , the QKP profit function reads as

$$f(S) = \sum_{j \in S} p_j + \sum_{i \in S} \sum_{j \in S, i < j} a_{ij} \tag{2}$$

and considers the pairwise interaction between couples of items. Observe that in this case the  $a_{ij}$  coefficients can take also negative values; thus, the maximum profit item set may be nonmaximal with respect to inclusion, which is a main difference with respect to KP.

QKP has been introduced in the eighties, see [11], and models many real-world applications. Though its simple definition, it is one of the most challenging optimization problems arising in practice, and it attracted a large amount of research. A survey

on heuristics, reduction techniques, branch-and-bound algorithms, and approximation results is given in [18], where some QKP applications are also mentioned. The interested reader is referred to [7, 14, 19] for further reading on the topic.

A further generalization of KP arises when considering the profit for item sets with cardinality larger than two. In this case, one is given a family  $T$  that contains all item sets with non-zero profit, and the profit for a given item set  $S \in T$  is defined as

$$f(S) = \sum_{S' \subseteq S} g(S') \quad (3)$$

where  $g(S')$  denotes the (possibly, negative) profit associated with each item set  $S' \in T$ . The profit of an item set  $S$  defined by (3) takes into account also the contribution of each proper subset  $S' \subset S$ .

There are, however, many applications in which the profit of an item set does not involve included subsets, i.e.,  $f(S)$  does not explicitly depend on the profit  $f(S')$  of all subsets  $S' \subset S$ . The resulting class of problems has been addressed in the Computational Social Choice literature, in the context of preference aggregation, voting theory, and ranking systems among others (see, e.g., [8]). We refer the interested reader to the seminal work of [5] where the paradox of multiple elections is studied. For example, consider a situation in which one has to select a number of members for a committee subject to a budget constraint, and the quality of the committee is related to both individual “scores” of the members, and to interrelations between subsets of members. It may happen that two members have high individual scores, but show some incompatibility that produces a negative impact on the quality of the committee, and the same situation may involve a larger number of members of the committee. This kind of situations have been introduced and studied in [23]. Thus, an optimal committee can be determined by taking into account the mutual scores of all subsets of candidate members that may reduce the total quality of the solution found. Similarly, the model above can be used in Multiagents Systems where the interest of some individual agents is conflicting with that of the overall system (see, [12]).

More in general, PKP can be used to model situations in which items are partitioned into two subsets, and a hard constraint imposes that the solution must include an even number of items from one subset. In this case, maximizing the product of the profits of the selected items is a proxy for guiding the solution to include an even number of items from that subset. In the context of Computational Social Choice, PKP models the special case of selection of committee members built starting from two groups of people, i.e., candidates that have to be taken in pairs and the others.

PKP also models a category of knapsack problems where one has to select among two different categories of items, say, “small” and “large” ones. Consider for example a special knapsack divided into compartments, where each compartment can accommodate either a large item or two small ones. Moreover, there are side constraints (e.g., stability when shipping fragile objects) imposing that in a feasible solution each compartment has to be either empty or full. This problem can be modelled as a PKP by assigning a negative profit to the small items while the knapsack constraint limits the total weight.

The *Product Knapsack Problem* considered in this paper is a special case of the KP with objective function (3). To the best of our knowledge, this is the first paper on this specific nonlinear variant of the KP, whereas a large amount of research has been devoted to 0–1 polynomial optimization. In particular, the most effective approaches either transform the problem at hand into a quadratic one (see, e.g., [6, 20]), or linearize each monomial in the objective function by adding new variables and constraints (see, e.g., [3, 17, 22]).

*Paper contribution.* In Sect. 2 we introduce possible integer nonlinear and linear programming formulations for PKP; for each integer linear formulation, we discuss how to solve the associated linear programming relaxation and the quality of the corresponding upper bound. In Sect. 3, we give an alternative approach based on a dynamic programming algorithm, and discuss about the complexity of PKP. A computational analysis on the presented models and algorithms on a large set of instances derived from the literature is given in Sect. 4. Finally, Sect. 5 draws some conclusions.

## 2 Mathematical formulations

In this section we first present a simple mixed integer nonlinear formulation of PKP. Then, we present a reduction of PKP to KP that can be applied under some hypotheses. Finally, we remove these hypotheses and consider the general version of PKP for which we introduce two alternative ILP formulations and study the associated linear programming relaxations.

### 2.1 Mixed integer nonlinear formulation

We introduce two variables for each item  $j \in N$ : a binary variable  $x_j$ , which takes value 1 if item  $j$  is selected and 0 otherwise, and a continuous variable  $y_j$ , which takes value  $p_j$  if item  $j$  is selected and 1 otherwise. Then, a mathematical (MINLP) model for PKP can be obtained as follows:

$$\max \prod_{j \in N} y_j \quad (4)$$

$$\sum_{j \in N} w_j x_j \leq C \quad (5)$$

$$y_j = 1 + (p_j - 1)x_j \quad j \in N \quad (6)$$

$$x_j \in \{0, 1\} \quad j \in N. \quad (7)$$

Objective function (4) maximizes the product of  $y$  variables. Constraints (6) define the correct value of each variable  $y_j$  ( $\forall j \in N$ ): if  $x_j = 1$ , variable  $y_j$  takes value  $p_j$  and item  $j$  contributes to the objective function with its profit. If instead  $x_j = 0$  then we have  $y_j = 1$ , i.e., item  $j$  gives no contribution to the solution profit. Observe that some entries in the objective function (4) can take negative values, hence  $y$  variables cannot be bounded to non-negative values. The model has  $2n$  variables (half binary,

half continuous) and  $n + 1$  linear constraints. The objective function is highly nonlinear as it includes the product of the  $y$  variables.

### 2.2 Reduction to KP

We present here a reduction of PKP to KP that is valid only in case all profits are positive, i.e.,  $p_j > 0 \quad \forall j \in N$ .

Indeed, observe that in this case we have that each  $y_j$  is strictly positive  $\forall j \in N$ . Thus, we can apply the log operator to objective function (4) and exploit the fact that  $\log \prod_{j \in N} y_j = \sum_{j \in N} \log y_j$ . This allows us to reformulate MINLP as a separable problem. Observing that  $\log y_j$  is zero if  $x_j = 0$ , and is equal to  $\log p_j$  otherwise, we can project the  $y$  variables out, and obtain the following reformulation

$$\max \left\{ \sum_{j \in N} (\log p_j) x_j : \sum_{j \in N} w_j x_j \leq C, x_j \in \{0, 1\}, j \in N \right\},$$

that is a standard KP in which each item  $j$  has a profit equal to  $\log p_j$ .

### 2.3 First ILP formulation

In this section, we present an ILP formulation for the case  $N^- \neq \emptyset$ . The ILP model is obtained by explicitly modeling Observation 1 by means of linear constraints, so as to avoid the distinction between items with positive and negative profit. In particular, considering the absolute values of the profits, one can apply the reduction to KP described in the previous section.

More in details, we introduce an additional variable

$$\alpha = \text{Number of pair of items } j \in N^- \text{ that are selected}$$

that is bounded to assume integer values, and obtain the following (ILP1) model:

$$\max \sum_{j \in N} (\log |p_j|) x_j \tag{8}$$

$$\sum_{j \in N} w_j x_j \leq C \tag{9}$$

$$\sum_{j \in N^-} x_j = 2\alpha \tag{10}$$

$$x_j \in \{0, 1\} \qquad j \in N \tag{11}$$

$$\alpha \in \mathbb{Z}_+. \tag{12}$$

In the model above, the left-hand-side of constraint (10) counts the number of items with negative profit that are selected. Since variable  $\alpha$  is integer, the effect of

this constraint is to impose this number to be even, which makes the resulting solution value positive, i.e., it allows us to have the absolute value of the profits in the objective function.

Let us denote by LP1 the Linear Programming (LP) relaxation of ILP1, obtained by replacing (11) and (12) by  $0 \leq x_j \leq 1 \quad \forall j \in N$  and  $\alpha \geq 0$ , respectively.

**Proposition 2** *An optimal solution to LP1 can be computed in linear time.*

*Proof* Dropping integrality requirement on  $\alpha$  makes constraint (10) immaterial. The remaining problem is the LP relaxation of KP, that can be solved in  $O(n)$  time finding the critical item in linear time (see [1]) and applying Dantzig’s algorithm [10]. Given an optimal solution to the LP relaxation of KP, the optimal value for variable  $\alpha$  can be computed a posteriori using (10). □

### 2.4 Second ILP formulation

An alternative ILP formulation for PKP can be obtained by modelling Observation 1 in a different way. Let us define the set  $J = \{(i, j) : i \in N^-, j \in N^-, i < j\}$  containing all distinct pairs  $(i, j)$  of items that have negative profit, and introduce a binary variable  $z_j$  for each item  $j \in N^+$  which takes value 1 if item  $j$  is selected and 0 otherwise. In addition, for each pair  $(i, j) \in J$ , we introduce a binary variable  $\beta_{ij}$  which takes value 1 if both items  $i$  and  $j$  are selected (and 0 otherwise). Then, a second (ILP2) model for PKP is as follows:

$$\max \sum_{j \in N^+} (\log p_j) z_j + \sum_{(i,j) \in J} (\log p_i p_j) \beta_{ij} \tag{13}$$

$$\sum_{j \in N^+} w_j z_j + \sum_{(i,j) \in J} (w_i + w_j) \beta_{ij} \leq C \tag{14}$$

$$\sum_{i:(i,k) \in J} \beta_{ik} + \sum_{i:(k,i) \in J} \beta_{ki} \leq 1 \quad k \in N^- \tag{15}$$

$$z_j \in \{0, 1\} \quad j \in N^+ \tag{16}$$

$$\beta_{ij} \in \{0, 1\} \quad (i, j) \in J. \tag{17}$$

Variables  $\beta$  are used to model the fact that items in  $N^-$  must be selected in pairs in any optimal solution; hence, each variable  $\beta_{ij}$  counts the profit (and weight) of simultaneously packing items  $i$  and  $j$ . Given that, we can remove the distinction between items with positive and negative profit and apply the reduction given in Sect. 2.2 to define objective function (13). Finally, constraints (15) impose that each item  $k \in N^-$  is paired with at most another item  $i \in N^-$ .

### 2.5 Comparison between the two ILP models

Observe that the number of variables of model ILP2 is  $|N^+| + |N^-|(|N^-| - 1)/2$ , i.e., it is quadratic in the number of items with negative profits—as opposed to model



ILP1 that has a linear size independently of the number of positive and negative items. Furthermore, the first model is smaller also in terms of number of constraints, 2 versus  $1 + |N^-|$ . However, the following observation shows that model ILP2 dominates model ILP1 in terms of LP relaxation. As before, let us denote by LP2 the Linear Programming (LP) relaxation of ILP2, obtained by replacing (16) and (17) by  $0 \leq z_j \leq 1 \quad \forall j \in N$  and  $0 \leq \beta_{ij} \leq 1 \quad \forall (i, j) \in J$ , respectively. In addition, let  $opt(LP1)$  and  $opt(LP2)$  denote the upper bounds produced by relaxations LP1 and LP2, respectively.

**Proposition 3** *LP2 dominates LP1. Moreover, the ratio  $\frac{opt(LP1)}{opt(LP2)}$  can be arbitrarily large.*

*Proof* We will show that any feasible solution  $(z^*, \beta^*)$  of LP2 can be converted into a feasible solution  $x^*$  to LP1 that has the same objective function value, and that the opposite is not always possible.

Let  $(z^*, \beta^*)$  be a feasible solution to LP2, and define a solution  $x^*$  as follows

$$x_k^* = \begin{cases} z_k^* & \text{if } k \in N^+; \\ \sum_{i:(i,k) \in J} \beta_{ik}^* + \sum_{i:(k,i) \in J} \beta_{ki}^* & \text{otherwise} \end{cases} \quad k \in N \tag{18}$$

First, observe that upper bounds on the  $z$  variables and constraints (15) ensure that no item is taken more than once. The total weight of the selected items is

$$\begin{aligned} \sum_{k \in N} w_k x_k^* &= \sum_{k \in N^+} w_k z_k^* + \sum_{k \in N^-} w_k \left( \sum_{i:(i,k) \in J} \beta_{ik}^* + \sum_{i:(k,i) \in J} \beta_{ki}^* \right) = \sum_{k \in N^+} w_k z_k^* \\ &\quad + \sum_{(i,k) \in J} (w_i + w_k) \beta_{ik}^* \end{aligned}$$

i.e., solution  $x^*$  is feasible by definition. Similarly, plugging  $x^*$  into (8), the solution profit is

$$\begin{aligned} \sum_{k \in N} (\log |p_k|) x_k^* &= \sum_{k \in N^+} (\log |p_k|) z_k^* + \sum_{k \in N^-} (\log |p_k|) \left( \sum_{i:(i,k) \in J} \beta_{ik}^* + \sum_{i:(k,i) \in J} \beta_{ki}^* \right) \\ &= \sum_{k \in N^+} (\log p_k) z_k^* + \sum_{(i,k) \in J} (\log p_i p_k) \beta_{ik}^* \end{aligned}$$

that is the same profit given by (13).

Consider now the following instance with  $n = 3$ , profits  $p = (-2M, -\frac{1}{M}, 2)$ , weights  $w = (1, 1, 1)$  and capacity  $C = 2$ , where  $M$  is a sufficiently large number. An optimal solution to LP1 is  $x_1 = 1, x_2 = 0, x_3 = 1$  and  $\alpha = 1/2$ , and has value  $\log 2M + \log 2$ , while an optimal solution of LP2 is  $z_3 = 1$  and  $\beta_{1,2} = 1/2$  and

has value  $3/2 \log 2$ . Thus, the ratio  $\frac{opt(LP1)}{opt(LP2)}$  is arbitrarily large for sufficiently large  $M$ . □

Though LP2 theoretically dominates LP1 (see Proposition 3), Proposition 4 hereinafter shows that instances for which  $opt(LP2) < opt(LP1)$  are very rare in practice, in particular when the number of items with negative profit is large.

**Proposition 4** *Let  $x^*$  be an optimal solution of LP1 and  $\bar{N}^- = \{j \mid j \in N^- \text{ and } x_j^* > 0\}$ . If  $|\bar{N}^-| \geq 3$ , then  $opt(LP1) = opt(LP2)$ .*

*Proof* Using Proposition 3, we have only to show that  $opt(LP2)$  cannot be smaller than  $opt(LP1)$ . To this aim, we construct a solution  $(z^*, \beta^*)$  that is feasible for LP2 and has the same value as solution  $x^*$ . Assume w.l.o.g. that  $N$  is sorted with items with negative profit first, i.e.,  $N^- = \{1, \dots, |N^-|\}$  and  $N^+ = \{|N^-|+1, \dots, |N|\}$ . Assume also that the negative-profit items and positive-profit items are sorted separately by non-increasing  $\frac{\log |p_j|}{w_j}$  values. Let  $s = |\bar{N}^-|$ ,  $t = s - 1$ ,  $u = s - 2$  and assume that  $s$  is odd. As observed in the proof of Proposition 2, the optimal solution of LP1 has the same structure as that of the continuous relaxation of the knapsack problem, i.e., all items  $j \in N^- : j < s$  (including  $t$  and  $u$ ) have  $x_j^* = 1$ . Define the following solution:

$$\begin{aligned} z_j^* &= x_j^* \quad \text{for each item } j \in N^+ \\ \beta_{j,j+1}^* &= 1 \quad \text{for } j = 1, 3, 5, \dots, s - 4 \\ \beta_{ut}^* &= 1 - \frac{x_s^*}{2}; \quad \beta_{us}^* = \frac{x_s^*}{2}; \quad \beta_{ts}^* = \frac{x_s^*}{2}, \end{aligned}$$

and all the remaining  $\beta$  variables are set to zero. It is easy to check that solution  $(z^*, \beta^*)$  is feasible for LP2 and its value is

$$\begin{aligned} &\sum_{j \in N^+} w_j z_j^* + \sum_{\substack{j=1 \\ j \text{ odd}}}^{s-4} \log(p_j p_{j+1}) \beta_{j,j+1}^* + \log(p_u p_t) \beta_{ut}^* \\ &\quad + \log(p_u p_s) \beta_{us}^* + \log(p_t p_s) \beta_{ts}^* \\ &= \sum_{j \in N^+} w_j x_j^* + \sum_{\substack{j=1 \\ j \text{ odd}}}^{s-4} (\log(p_j) x_j^* + \log(p_{j+1}) x_{j+1}^*) + \log(p_u p_t) \left(1 - \frac{x_s^*}{2}\right) \\ &\quad + \log(p_u p_s) \frac{x_s^*}{2} + \log(p_t p_s) \frac{x_s^*}{2} \\ &= \sum_{j \in N^+} w_j x_j^* + \sum_{\substack{j=1 \\ j \text{ odd}}}^{s-4} (\log(p_j) x_j^* + \log(p_{j+1}) x_{j+1}^*) + \log(p_s) x_s^* \\ &\quad + \log(p_t) x_t^* + \log(p_u) x_u^*. \end{aligned}$$

Similarly, if  $s$  is even, let  $v = s - 3$  and define the following solution

$$\begin{aligned}
 z_j^* &= x_j^* \quad \text{for each item } j \in N^+ \\
 \beta_{j,j+1}^* &= 1, \quad j = 1, 3, 5, \dots, s - 5; \\
 \beta_{ts}^* &= x_s^*; \beta_{ut}^* = \frac{1 - x_s^*}{2}; \quad \beta_{vt}^* = \frac{1 - x_s^*}{2}; \quad \beta_{vu}^* = \frac{1 + x_s^*}{2}.
 \end{aligned}$$

In this case too, all the remaining  $\beta$  variables are set to zero. The value of this solution is

$$\begin{aligned}
 &\sum_{j \in N^+} w_j z_j^* + \sum_{\substack{j=1 \\ j \text{ odd}}}^{s-5} \log(p_j p_{j+1}) \beta_{j,j+1}^* + \log(p_t p_s) \beta_{ts}^* + \log(p_u p_t) \beta_{ut}^* \\
 &\quad + \log(p_v p_t) \beta_{vt}^* + \log(p_v p_u) \beta_{vu}^* \\
 &= \sum_{j \in N^+} w_j x_j^* + \sum_{\substack{j=1 \\ j \text{ odd}}}^{s-5} (\log(p_j) x_j^* + \log(p_{j+1}) x_{j+1}^*) + \log(p_t p_s) x_s^* \\
 &\quad + \log(p_u p_t) \frac{1 - x_s^*}{2} + \log(p_v p_t) \frac{1 - x_s^*}{2} + \log(p_v p_u) \frac{1 + x_s^*}{2} \\
 &= \sum_{j \in N^+} w_j x_j^* + \sum_{\substack{j=1 \\ j \text{ odd}}}^{s-5} (\log(p_j) x_j^* + \log(p_{j+1}) x_{j+1}^*) + \log(p_s) x_s^* + \log(p_t) x_t^* \\
 &\quad + \log(p_u) x_u^* + \log(p_v) x_v^*.
 \end{aligned}$$

Thus, the solution has the same profit as  $x^*$ , which concludes the proof. □

A similar argument shows that  $opt(LP1) = opt(LP2)$  holds also in case  $|\bar{N}^-| = 2$  provided that  $x_j^* = 1 \forall j \in N^-$ , i.e., the optimal solution of LP1 fully packs exactly two items with negative profit. Therefore, it is very unlikely to generate instances which do not satisfy the requirements of Proposition 4.

We conclude this section by observing that model ILP2 can be viewed as a 0-1 Knapsack Problem with Conflict Graph (KGC) with  $|N^+| + |J|$  items. Objective function (13) and constraints (14), (16), and (17) can be rewritten as a standard KP, while (15) are the conflict constraints, see [24]. The conflict constraints can be represented via a conflict graph, with one node for each item in the knapsack and one edge for each couple of items that can not be taken together in a feasible solution.

The results presented in [4] show that, from a computational point of view, the density (i.e., the ratio between the number of edges and the the cardinality of the edge set of the complete graph having the same number of vertices) of the associated conflict graph can impact the performances of the solution method used to solve one instance of KGC. In our case, the density of the conflict graph can be computed with a closed formula that depends only on  $|N^+|$  and  $|N^-|$ . The conflict graph contains

$$|N^+| + |J| = |N^+| + \frac{|N^-|(|N^-| - 1)}{2} \text{ nodes.}$$

Since there are  $|N^-|$  constraints (15),

each involving  $|N^-| - 1$  variables, it is easy to see that the total number of edges is  $\frac{|N^-|(|N^-| - 1)(|N^-| - 2)}{2}$ . As  $|N^+| = O(n)$  and  $|N^-| = O(n)$ , the overall density of the graph is  $O(1/n)$ , i.e., the resulting conflict graph is sparse for sufficiently large instances. It has been computationally proved in [4, 21] that, for sparse graphs, the direct use of an ILP solver is the most efficient way to solve KGC, whereas more efficient combinatorial methods exist for dense graphs.

### 3 Dynamic Programming and computational complexity

Dynamic Programming (DP) was introduced by Bellman [2]. Subsequently, several authors proposed DP algorithms for KP (see for example [13] or [15]). In this section we first present a dynamic programming for PKP and then we determine the computational complexity of the problem.

Our DP algorithm (called  $DP_{PKP}$ ) is composed by  $n$  steps. At each step  $j$ , the algorithm computes, for each capacity value  $s = 0, \dots, C$ , both the most positive and the most negative PKP objective function values that can be obtained using the first  $j$  items with a capacity equal to  $s$ . All these values can be stored into two matrices  $f_j[s]$  and  $g_j[s]$ , and can be computed using the following recursive formulas:

$$f_j[s] = \begin{cases} \max\{f_{j-1}[s], p_j \cdot f_{j-1}[s - w_j], p_j \cdot g_{j-1}[s - w_j]\} & \text{if } s \geq w_j; \\ f_{j-1}[s] & \text{otherwise} \end{cases} \quad (19)$$

$$g_j[s] = \begin{cases} \min\{g_{j-1}[s], p_j \cdot g_{j-1}[s - w_j], p_j \cdot f_{j-1}[s - w_j]\} & \text{if } s \geq w_j; \\ g_{j-1}[s] & \text{otherwise} \end{cases} \quad (20)$$

with the initialization  $f_0[s] = 1$  and  $g_0[s] = 1, \forall s = 0, \dots, C$ . The optimal PKP solution value is finally given by  $f_n[C]$ .

**Proposition 5**  $DP_{PKP}$  computes an optimal solution value of PKP in  $O(nC)$  time.

*Proof* The time complexity is obvious, as  $O(nC)$  entries have to be computed, each requiring  $O(1)$  time. We now prove that the algorithm computes an optimal solution by induction. Assume that  $f_{j-1}$  and  $g_{j-1}$  have been correctly computed for each capacity value  $s$ ; consider now a capacity value  $s$  and evaluate the possible update of the associated  $f_j$  entry—the proof is identical for what concerns the  $g_j$  entry. If  $s < w_j$  then item  $j$  cannot be inserted in any feasible solution, hence no update can take place. Otherwise, either the new item is not packed in the partial solution, keeping  $f_j[s]$  unchanged, or item  $j$  is selected. In this latter case, if  $p_j > 0$  the new value of  $f_j[s]$  is obtained multiplying  $p_j$  by the most positive objective function value obtained with a partial capacity decreased by  $w_j$ , i.e.,  $p_j \cdot f_{j-1}[s - w_j]$ . If instead  $p_j < 0$ , a maximum profit solution that packs item  $j$  is obtained multiplying  $p_j$  by the most negative objective function value obtained with a partial capacity decreased by  $w_j$ , i.e.,  $p_j \cdot g_{j-1}[s - w_j]$ . In both cases, the update is implemented only in case it is profitable.  $\square$

To illustrate the algorithm, Table 1 reports matrices  $f$  and  $g$  for the following numerical example:  $n = 4$  (four items), profits  $p = (5, -3, -4, 2)$ , weights  $w =$

**Table 1** States of algorithm  $DP_{PKP}$  for the example instance of four objects and capacity  $C = 3$

f		j				
		0	1	2	3	4
s	0	1	1	1	1	1
	1	1	1	1	1	2
	2	1	5	5	12	12
	3	1	5	5	12	24
g		j				
		0	1	2	3	4
s	0	1	1	1	1	1
	1	1	1	-3	-4	-4
	2	1	1	-3	-4	-8
	3	1	1	-15	-20	-20

**Fig. 1** Dynamic programming algorithm  $DP_{PKP}$

```

Algorithm  $DP_{PKP}$ :
initialization
for  $s := 0$  to  $C$  do
     $f[s] = 1; g[s] = 1$ 
consider one item at a time
for  $j := 1$  to  $n$  do
    for  $s := C$  down to  $w_j$  do
        if  $p_j \geq 0$  then
            if  $(p_j \cdot f[s - w_j] > f[s])$  then  $f[s] = p_j \cdot f[s - w_j]$ 
            if  $(p_j \cdot g[s - w_j] < g[s])$  then  $g[s] = p_j \cdot g[s - w_j]$ 
        else
            if  $(p_j \cdot g[s - w_j] > f[s])$  then  $f[s] = p_j \cdot g[s - w_j]$ 
            if  $(p_j \cdot f[s - w_j] < g[s])$  then  $g[s] = p_j \cdot f[s - w_j]$ 
    return  $f[C]$ .
    
```

(2, 1, 1, 1) and capacity  $C = 3$ . Entry  $f_4[3]$  provides the optimal solution value equal to 24.

A possible implementation of the algorithm is given in Fig. 1. In this scheme, instead of storing two  $n \cdot (C + 1)$  matrices for the  $f$  and  $g$  entries, we use two  $C + 1$  vectors. For each item  $j$ , these vectors are scanned according to decreasing values of the available capacity  $s$ , possibly updating the associated entries. This implementation allows to reduce the memory requirement of the algorithm, yielding some improvements from a computational viewpoint.

Finally, it is important to remark that an equivalent version of Algorithm  $DP_{PKP}$  can be obtained by storing in  $f$  and  $g$  the logarithm of the (absolute values of the) products showed in equations (19) and (20). This a crucial step for the practical applicability of the algorithm (see, Sect. 4).

**Proposition 6** *PKP is weakly NP-hard.*

*Proof* Given the dynamic programming algorithm above, PKP cannot be strongly NP-hard. We now show that any instance  $I$  of KP can be polynomially transformed into an instance  $I'$  of PKP that has the same solution. Let  $I = (n, C, p, w)$  be a KP instance, and define  $I'$  as follows:  $n' = n, C' = C, p'_j = \log^{-1} p_j$  and  $w'_j = w_j$

$\forall j$ . Indeed, all profits in  $I'$  are positive and we can apply the reduction introduced in Sect. 2.2, which produces an optimal solution to a KP instance with profits  $p'_j = \log \log^{-1} p_j = p_j$ , i.e., the same profits as in instance  $I$ .  $\square$

## 4 Computational experiments

The primary goal of this computational study is to compare the performances of the ILP models discussed in Sect. 2 with the Dynamic Programming algorithm ( $DP_{\text{PKP}}$ ) described in Sect. 3. A second goal of this section is to determine the size of the instances that can be solved to proven optimality within short computing time.

All the experiments have been performed on a computer with a 3.40 Ghz 8-core Intel Core i7-3770 processor and 16Gb RAM, running a 64 bits Linux operating system. All the codes were compiled with g++ (version 4.8.4) using `-O3` as optimization option. We have tested the ILP formulations using `Cplex 12.6.0`, in single-thread mode. All `Cplex` parameters are at their default values except the optimality tolerances which have been set to 0 to allow a meaningful comparison with the solutions produced by the DP algorithm.

*Instances:* The PKP is a new problem and, accordingly, no benchmark instances are available in the literature. Nevertheless, since a PKP instance can be easily obtained from a KP instance by introducing items with negative profits, we create a PKP benchmark starting from nine classes of KP instances of the literature. The KP item profits and weights are obtained using the random generator proposed in [15] (available at <http://www.diku.dk/~pisinger/codes.html>) with the following features, where u.r. stands for “uniformly random integer”:

1. *Uncorrelated:*  $w_j$  u.r. in  $[1, \bar{R}]$ ,  $p_j$  u.r. in  $[1, \bar{R}]$ .
2. *Weakly correlated:*  $w_j$  u.r. in  $[1, \bar{R}]$ ,  $p_j$  u.r. in  $[\max\{1, w_j - \bar{R}/10\}, w_j + \bar{R}/10]$ .
3. *Strongly correlated:*  $w_j$  u.r. in  $[1, \bar{R}]$ ,  $p_j = w_j + \bar{R}/10$ .
4. *Inverse strongly correlated:*  $p_j$  u.r. in  $[1, \bar{R}]$ ,  $w_j = p_j + \bar{R}/10$ .
5. *Almost strongly correlated:*  $w_j$  u.r. in  $[1, \bar{R}]$ ,  $p_j$  u.r. in  $[w_j + \bar{R}/10 - \bar{R}/500, w_j + \bar{R}/10 + \bar{R}/500]$ .
6. *Subset-sum:*  $w_j$  u.r. in  $[1, \bar{R}]$ ,  $p_j = w_j$ .
7. *Even-odd subset-sum:*  $w_j$  even value u.r. in  $[1, \bar{R}]$ ,  $p_j = w_j$ ,  $C$  odd.
8. *Even-odd strongly correlated:*  $w_j$  even value u.r. in  $[1, \bar{R}]$ ,  $p_j = w_j + \bar{R}/10$ ,  $C$  odd.
9. *Uncorrelated with similar weights:*  $w_j$  u.r. in  $[100\bar{R}, 100\bar{R} + \bar{R}/10]$ ,  $p_j$  u.r. in  $[1, \bar{R}]$ .

For a given class, a given number of items  $n$ , and a specific value of  $\bar{R}$ , the capacity  $C$  is defined as a percentage of the total item weight ( $W = \sum_{j=1}^n w_j$ ) as follows:  $C = \lfloor \bar{c} \cdot W \rfloor$ , where  $\bar{c}$  is another instance-generator parameter. Moreover,  $C$  is increased by 1, if even, for classes 7 and 8. Given a KP instance, we define a PKP problem using an additional parameter  $\bar{m}$ , that represents the percentage of items with negative profit. In particular, we set  $\bar{n} = \lfloor \bar{m} \cdot n \rfloor$  and change the sign of the profit for  $\bar{n}$  randomly selected items.

Our test bed is divided in two sets. The first set (called set A) consists of instances of small size obtained using the following set of instance-generator parameters:  $\bar{R} = 1000$ ;  $n \in \{50, 100, 500, 1000\}$ ;  $\bar{c} = 0.5$ ; and  $\bar{m} \in \{0.66, 0.50, 0.25\}$ . For each combination of these parameters, one problem is generated for each of the 9 aforementioned classes, thus producing a test bed with 108 instances. The second set (called set B) consists of instances of larger size, and is obtained using the following set of instance-generator parameters:  $\bar{R} \in \{100, 1000, 10000\}$ ;  $n \in \{500, 1000, 2000\}$ ;  $\bar{c} \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6\}$ ; and  $\bar{m} \in \{0.66, 0.50, 0.25\}$ . In this case too, one problem is generated for each combination of the parameters and for each class of KP instances, yielding a benchmark with 1458 problems. Set A is used to compare the performances of the two ILP models, while set B is used to compare the best-performing ILP model against the DP algorithm.

The test bed has been thought to measure the impact on the computational performances of the principal features of a PKP instance, i.e., the instance class, the number of items  $n$ , the capacity  $C$ , and the percentage of negative-profit items  $\bar{m}$ . It is worth mentioning that items in class 9 have, on average, weights that are 2 orders of magnitude larger than in the other classes. Since  $C$  is defined as a percentage of the sum of all item weights, and the time complexity of DP is  $O(nC)$ , these instances are specifically designed to be hard for the DP algorithm. Similarly, increasing the value of  $\bar{R}$  produces instances with larger  $W$  and, hence, harder for the DP algorithm. Preliminary computational experiments showed that the direct application of MINLP solvers to model (4)–(7) is not a viable way for solving these sets of instances. In most of the cases, the optimal solution includes many items, which makes the product of the associated profits (i.e., the objective function value) too large for being stored in the memory of a computer. For this reason, we do not present results for this approach. All the instances considered in our computational analysis are available at the following address: <http://or.dei.unibo.it/library/product-knapsack-problem>.

*Comparison between ILP models.* We now compare the computational performances of ILP1 and ILP2, i.e., the models presented in Sects. 2.3 and 2.4, on the instances of set A.

Table 2 is horizontally divided in three parts according to the 3 values taken by  $\bar{m}$ , i.e., the percentage of items with negative profit. Each line is associated with instances having the same number  $n$  of items, and each entry reports the aggregated results concerning 9 instances, one for each class. For both ILP1 and ILP2, we report the average CPU time (in seconds) required by `Cplex` to solve the model ( $t_{ILP}$ ) and to solve its linear programming relaxation ( $t_{LP}$ ). We also report the average number of explored nodes in the Branch-and-Cut tree and the size of the model in terms of number of variables (vars) and constraints (cons). In all tests, a time limit of 600 seconds is imposed. For model ILP2, some of the instances are not solved to proven optimality within the time limit; thus, we also report the number of unsolved instances (column T.L.). The average CPU time and number of nodes are computed considering only instances solved within the time limit. As already observed, ILP2 requires  $|N^+| + |N^-|(|N^-| - 1)/2$  binary variables, i.e., the size of the formulation grows quadratically with respect to the number of items and may become large in practice, which yields to computationally demanding models. In particular, ILP2 behaves poorly on average on this set of instances, and is unable to solve any instance with  $n = 1000$  items within

**Table 2** Computational comparison between ILP1 and ILP2 for instances set A

Instances	ILP1					ILP2							
	$\bar{n}$	$n$	gap	$t_{ILP}$	nodes	$t_{LP}$	vars	cons	T.L.	nodes	$t_{LP}$	vars	cons
0.66	50	50	1.21	0.01	15	0.00	51	2	2	1087432	0.00	545	34
	100	100	0.39	0.01	40	0.00	101	2	7	623436	0.00	2244	68
	500	500	0.07	0.04	133	0.00	501	2	9	-	0.33	55445	334
	1000	1000	0.03	0.04	30	0.00	1001	2	9	-	6.12	222444	668
0.50	50	50	1.19	0.00	0	0.00	51	2	0	6881	0.00	325	26
	100	100	0.38	0.02	34	0.00	101	2	6	4231	0.00	1275	51
	500	500	0.07	0.03	114	0.00	501	2	8	7526	0.15	31375	251
	1000	1000	0.03	0.04	57	0.00	1001	2	9	-	2.05	125250	501
0.25	50	50	1.22	0.01	9	0.00	51	2	0	811	0.00	115	14
	100	100	0.39	0.03	274	0.00	101	2	2	742954	0.00	375	26
	500	500	0.07	0.03	189	0.00	501	2	4	81961	0.02	8125	126
	1000	1000	0.04	0.05	283	0.00	1001	2	9	-	0.15	31875	251



10 minutes of CPU time. In addition, ILP2 is not capable of solving several instances with  $n \leq 500$ . On the contrary, ILP1 is able to solve all the instances in a fraction of a second. For this reason, we decided to exclude ILP2 in the following analysis.

It is worth mentioning that, though LP2 is theoretically stronger than LP1, the two models always provide the same upper bound for the instances in set A. Strict dominance applies only for instances with specific characteristics (see Sect. 2.5) and it never happens in our test bed. For this reason, in Table 2 we report only one LP gap for both models (computed as  $\text{gap} = 100 \cdot \frac{\text{opt}(LP) - \text{opt}}{\text{opt}(LP)}$  where  $\text{opt}(LP)$  is the optimal solution value of the linear programming relaxation, and  $\text{opt}$  is the optimal solution value).

Moreover, thanks to an additional set of preliminary experiments, we noticed that LP2 is stronger in practice only for instances where the percentage of items with negative profit  $\bar{m}$  is very small, namely for  $\bar{m} < 0.1$ .

Finally, we observe that the LP gap decreases when the number of items increases. This is not surprising as, given a solution of LP1, one can derive a feasible integer solution by removing at most two items, namely the critical item (if any) and possibly one item with negative profit. As the gap is related to the contribution of at most two items, its percentage value decreases when feasible solutions include many items (as in the case in which  $n$  is large), similar to what happens to the LP relaxation of the knapsack problem. In addition, we note that the number of nodes explored by ILP2 is always several orders of magnitudes larger than the same figure for ILP1.

Since the solution of LP2 requires considerably more time than LP1, a future line of research could be devoted to the development of a branch-and-price algorithm, generating binary variables on the fly, to reduce the overall CPU time of the associated relaxation.

*Comparison between ILP1 and algorithm  $DP_{PKP}$ .* Tables 3 and 4 report the average CPU times required by algorithm  $DP_{PKP}$  and by `Cplex` for model ILP1 in order to solve to proven optimality the instances of set B. A time limit of 600 seconds is imposed for each run; as for Table 2, we report the number of instances that are not solved to proven optimality (columns T.L.). Table 3 is divided in 9 blocks, one for each class of instances. Each block is horizontally divided in three parts according to the 3 values of  $\bar{R}$ , and each line is associated with instances that share the same number  $n$  of items. Each entry reports the average CPU time of the instances solved out of 18 instances, i.e., grouping together the different values of the parameters  $\bar{c}$  and  $\bar{m}$ . The results show that algorithm  $DP_{PKP}$  is able to solve to proven optimality all the instances of the first 8 classes, but fails on some problems of class 9. The CPU time of the DP algorithm is directly proportional to the value  $\bar{R}$  and  $n$ , which is not surprising as its computational complexity is  $O(nC)$ . On the other hand, the computational performance of ILP1 is less predictable, i.e., for many classes of instances there are several problems for which ILP1 hits the time limit. Nevertheless for instances of classes 1 and 9, ILP1 dominates algorithm  $DP_{PKP}$ . For the instances with  $\bar{R} = 100$  and classes from 2 to 8 the DP algorithm almost always outperforms ILP1. The instances of classes 3 and 8 are the most difficult ones for ILP1, with 6 and 5 unsolved instances, respectively.

Table 4 reports additional results aimed at evaluating the effect of the capacity  $C$  on the computational performances of ILP1 and algorithm  $DP_{PKP}$ . Since the value

**Table 3** Computational comparison between ILP1 and the DP algorithm for instance set B

$\bar{R}$	$n$	Class 1			Class 2			Class 3					
		$t_{DP}$	T.L.	$t_{ILP1}$	T.L.	$t_{ILP1}$	T.L.	$t_{ILP1}$	T.L.	$t_{ILP1}$	T.L.		
100	500	0.02	0	0.03	0	0.02	0	0.07	0	0.02	0	0.01	1
	1000	0.07	0	0.05	0	0.07	0	0.03	0	0.07	0	0.01	2
	2000	0.27	0	0.07	0	0.26	0	0.57	0	0.26	0	0.01	2
	500	0.15	0	0.03	0	0.15	0	0.03	0	0.16	0	0.03	0
	1000	0.59	0	0.05	0	0.64	0	0.08	0	0.65	0	0.03	0
	2000	2.95	0	0.06	0	2.92	0	6.05	1	2.92	0	0.07	0
10000	500	1.47	0	0.05	0	1.46	0	0.09	0	1.43	0	0.03	0
	1000	5.99	0	0.08	0	5.94	0	0.06	0	6.00	0	0.06	0
	2000	25.36	0	0.09	0	27.67	0	0.09	0	25.29	0	0.05	1
$\bar{R}$	$n$	Class 4			Class 5			Class 6					
		$t_{DP}$	T.L.	$t_{ILP1}$	T.L.	$t_{ILP1}$	T.L.	$t_{ILP1}$	T.L.	$t_{ILP1}$	T.L.		
100	500	0.02	0	0.06	0	0.02	0	0.01	0	0.02	0	0.01	1
	1000	0.08	0	1.34	0	0.07	0	10.49	0	0.07	0	0.44	0
	2000	0.33	0	0.06	0	0.28	0	0.01	1	0.30	0	0.01	0
1000	500	0.18	0	0.06	0	0.15	0	0.03	0	0.15	0	0.02	0
	1000	0.77	0	0.07	0	0.60	0	0.03	1	0.61	0	0.02	0
	2000	3.49	0	0.37	0	2.86	0	0.06	0	2.79	0	0.03	0

Table 3 continued

$\bar{R}$	$n$	Class 4			Class 5			Class 6					
		$t_{DP}$	T.L.	$t_{LLP1}$	T.L.	$t_{LLP1}$	T.L.	$t_{LLP1}$	T.L.	$t_{LLP1}$	T.L.		
10000	500	1.72	0	0.07	0	1.46	0	0.04	0	1.44	0	0.02	0
	1000	7.12	0	0.13	2	5.94	0	0.09	0	5.93	0	0.09	0
	2000	30.54	0	0.09	0	25.25	0	0.11	0	25.23	0	0.04	0
$\bar{R}$	$n$	Class 7			Class 8			Class 9					
		$t_{DP}$	T.L.	$t_{LLP1}$	T.L.	$t_{LLP1}$	T.L.	$t_{LLP1}$	T.L.	$t_{LLP1}$	T.L.		
100	500	0.02	0	0.00	0	0.02	0	0.00	0	2.95	0	0.03	0
	1000	0.07	0	0.01	1	0.07	0	0.01	1	11.91	0	0.04	0
	2000	0.29	0	2.18	0	0.28	0	0.01	1	54.20	0	0.56	3
1000	500	0.16	0	0.56	0	0.15	0	0.60	2	29.87	0	0.02	0
	1000	0.59	0	0.02	0	0.63	0	0.02	0	123.16	0	0.03	0
	2000	2.77	0	0.19	1	2.77	0	8.45	1	343.36	6	0.27	0
10000	500	1.42	0	0.02	0	1.43	0	0.02	0	297.57	0	0.03	0
	1000	5.91	0	14.11	0	5.94	0	10.63	0	330.42	15	0.03	0
	2000	25.93	0	0.06	0	26.42	0	0.06	0	--	18	0.18	0

**Table 4** Computational comparison between ILPI and the DP algorithm for instances with  $\bar{R} = 100$  and different capacity percentages

Class	$DP_{EXP}$				ILPI				T.L.	$\bar{c} = 0.8$	T.L.	$\bar{c} = 0.6$	T.L.	$\bar{c} = 0.4$	T.L.	$\bar{c} = 0.2$	T.L.	$\bar{c} = 0.8$	T.L.	
	$\bar{c} = 0.2$	$\bar{c} = 0.4$	$\bar{c} = 0.6$	$\bar{c} = 0.8$	$\bar{c} = 0.2$	$\bar{c} = 0.4$	$\bar{c} = 0.6$	$\bar{c} = 0.8$												
1	0.06	0.14	0.20	0.26	0.05	0.04	0	0	0	0.07	0	0.05	0	0	0	0	0	0	0	0
2	0.06	0.13	0.20	0.26	0.08	0.06	0	0	0	0.94	0	0.04	0	0	0	0	0	0	0	0
3	0.06	0.12	0.20	0.27	0.01	0.01	1	0	0	0.01	2	0.01	0	0	0	0	0	0	0	2
4	0.07	0.15	0.23	0.33	0.01	0.13	0	0	0	0.01	0	0.03	0	0	0	0	0	0	0	0
5	0.06	0.13	0.20	0.26	0.01	0.01	0	0	0	0.01	1	0.01	0	0	0	0	0	0	0	0
6	0.06	0.15	0.20	0.26	0.88	0.01	0	0	0	0.01	1	0.01	0	0	0	0	0	0	0	0
7	0.06	0.14	0.20	0.26	0.01	0.01	0	0	0	0.00	0	22.65	0	0	0	0	0	0	0	0
8	0.06	0.13	0.20	0.26	0.00	0.01	0	0	0	0.01	0	0.19	0	0	0	0	0	0	0	1
9	11.57	23.61	36.25	49.91	0.85	0.07	0	0	0	0.09	3	0.03	0	0	0	0	0	0	0	0

of the capacity depends on two parameters,  $\bar{R}$  and  $\bar{c}$ , we decided to include in this analysis only instances with the same value of  $\bar{R}$ , namely  $\bar{R} = 100$ , and to report results for different values of  $\bar{c}$ . Each line of the table is associated with a class of problems, thus entries report aggregated results for the 9 instances obtained with different combinations of parameters  $n$  and  $\bar{m}$ . We report the results associated with instances that have  $\bar{c} \in \{0.2, 0.4, 0.6, 0.8\}$ , i.e., some problems of the instance set B plus some additional instances with  $\bar{c} = 0.8$ . As for the previous tables, we report the average computing time and, for ILP1, the number of instances that could not be solved to proven optimality within the 600-second time limit. These results clearly show that increasing the capacity produces harder instances for algorithm  $DP_{PKP}$ , for which the associated CPU time grows almost linearly with parameter  $\bar{c}$ . Indeed, while the performances of algorithm  $DP_{PKP}$  are strongly dependent on  $\bar{R}$  and  $n$ , the behaviour of ILP1 is less predictable: out of the 22 instances that are not solved to optimality within the time limit, 4 involve 500 items only. In addition, we notice that classes 3 and 8 are the most challenging for ILP1. As to algorithm  $DP_{PKP}$ , classes from 1 to 8 are equally hard while, as expected, instances of class 9 are instead much harder.

## 5 Conclusions

In this article we study the PKP which is a variant of the Knapsack Problem in which the objective function optimizes the product of the profits of the selected items. The PKP is particularly interesting in the context of Computation Social Choice since it is a step in the direction of measuring the interaction among the members of a committee. We have demonstrated the computational complexity of the PKP and proposed different linear and nonlinear mathematical formulations. We have introduced a Dynamic Programming algorithm with a pseudo polynomial computational complexity. Finally, an extensive analysis on a large set of instances with diverse characteristics has computationally demonstrated that the Integer Linear Programming model ILP1 and the new DP algorithm proposed are the best performing exact approaches for the PKP.

With this manuscript we hope to stimulate the research in the boundaries of Optimization and Computation Social Choice. An interesting line of research in this direction would be the analysis of others objective functions which take into consideration different kinds of interrelation between the selected members of a committee.

**Acknowledgements** The authors would like to thank two anonymous referees for their helpful comments and Cecilia Bruni for her linguistic assistance.

## References

1. Balas, E., Zemel, E.: An algorithm for large zero-one knapsack problems. *Oper. Res.* **28**, 1130–1154 (1980)
2. Bellman, R.: *Dynamic Programming*. Princeton University Press, Princeton (1957)
3. Belotti, P., Lee, J., Liberty, L., Margot, F., Wächter, A.: Branching and bounds tightening techniques for non-convex MINLP. *Optim. Methods Softw.* **24**, 597–634 (2009)

4. Bettinelli, A., Cacchiani, V., Malaguti, E.: A branch-and-bound algorithm for the knapsack problem with conflict graph. Technical report (2016)
5. Brams, J.S., Kilgour, M.D., Zwicker, S.W.: The paradox of multiple elections. *Soc. Choice Welf.* **15**(2), 211–236 (1998)
6. Buchheim, C., Rinaldi, G.: Efficient reduction of polynomial zero-one optimization to the quadratic case. *SIAM J. Optim.* **18**(4), 1398–1413 (2007)
7. Caprara, A., Pisinger, D., Toth, P.: Exact solution of the quadratic knapsack problems. *INFORMS J. Comput.* **11**, 125–137 (1998)
8. Chevaleyre, Y., Endriss, U., Lang, J., Maudet, N.: A short introduction to computational social choice. In: van Leeuwen, J., Italiano, G.F., van der Hoek, W., Meinel, C., Sack, H., Plášil, F. (eds.) *SOFSEM 2007: Theory and Practice of Computer Science. SOFSEM 2007. Lecture Notes in Computer Science*, vol 4362. Springer, Berlin, Heidelberg (2007)
9. D'Ambrosio, C., Martello, S.: Heuristic algorithms for the general nonlinear separable knapsack problem. *Comput. Oper. Res.* **38**(2), 505–513 (2011)
10. Dantzig, G.: Discrete variable extremum problems. *Oper. Res.* **5**, 266–277 (1957)
11. Gallo, G., Hammer, P., Simeone, B.: Quadratic knapsack problems. *Math. Program. Study* **12**, 132–149 (1980)
12. Hao, J., Leung, H.: Interactions in Multiagent Systems: Fairness, Social Optimality and Individual Rationality, chap. Fairness in Cooperative Multiagent Systems. Springer, Berlin (2016)
13. Horowitz, E., Sahni, S.: Computing partitions with applications to the knapsack problem. *J. ACM* **21**(2), 277–292 (1974)
14. Kellerer, H., Pferschy, U., Pisinger, D.: *Knapsack Problems*. Springer, Berlin (2004)
15. Martello, S., Pisinger, D., Toth, P.: Dynamic programming and strong bounds for the 0–1 knapsack problem. *Manag. Sci.* **45**(3), 414–424 (1999)
16. Martello, S., Toth, P.: *Knapsack Problems: Algorithms and Computer Implementations*. Wiley, Chichester (1990)
17. Misener, R., Floudas, C.A.: ANTIGONE: algorithms for continuous/integer global optimization of nonlinear equations. *J. Glob. Optim.* **59**(2), 503–526 (2014)
18. Pisinger, D.: The quadratic knapsack problem—a survey. *Discrete Appl. Math.* **155**, 623–648 (2007)
19. Pisinger, D., Rasmussen, A., Sandvik, R.: Solution of large quadratic knapsack problems through aggressive reduction. *INFORMS J. Comput.* **19**, 280–290 (2007)
20. Rosenberg, I.G.: Reduction of bivalent maximization to the quadratic case. *Cahiers du Centre d'Etudes de Recherche Opérationnelle* **17**, 71–74 (1975)
21. Sadykov, R., Vanderbeck, F.: Bin packing with conflicts: a generic branch-and-price algorithm. *INFORMS J. Comput.* **25**, 244–255 (2013)
22. Tawarmalani, M., Sahinidis, N.V.: A polyhedral branch-and-cut approach to global optimization. *Math. Program.* **103**, 225–249 (2005)
23. Uckelman, J.: Alice and bob will fight: the problem of electing a committee in the presence of candidate interdependence. *Front. Artif. Intell. Appl.* **215**, 1023–1024 (2010)
24. Yamada, T., Kataoka, S., Watanabe, K.: Heuristic and exact algorithms for the disjunctively constrained knapsack problem. *Inf. Process. Soc. Jpn. J.* **43**, 2864–2870 (2002)