Integer Optimization with Penalized Fractional Values: The Knapsack Case

Enrico Malaguti[1], Michele Monaci[1], Paolo Paronuzzi[1], and Ulrich Pferschy[2]

[1] *DEI, University of Bologna, Viale Risorgimento 2, I-40136 Bologna, Italy.*
{enrico.malaguti, michele.monaci, paolo.paronuzzi}@unibo.it

[2] *Department of Statistics and Operations Research, University of Graz, Universitaetsstrasse 15, A-8010 Graz, Austria.* pferschy@uni-graz.at

Abstract

We consider integer optimization problems where variables can potentially take fractional values, but this occurrence is penalized in the objective function. This general situation has relevant examples in scheduling (preemption), routing (split delivery), cutting and telecommunications, just to mention a few. However, the general case in which variables integrality can be relaxed at cost of introducing a general penalty was not discussed before. As a case study, we consider the possibly simplest combinatorial optimization problem, namely the classical Knapsack Problem. We introduce the Fractional Knapsack Problem with Penalties (FKPP), a variant of the knapsack problem in which items can be split at the expense of a penalty depending on the fractional quantity. We analyze relevant properties of the problem, present alternative mathematical models, and analyze their performance from a theoretical viewpoint. In addition, we introduce a Fully Polynomial Time Approximation Scheme for the approximate solution of the general problem, and an improved dynamic programming approach that computes the optimal solution in one relevant case. We computationally test the proposed models and algorithms on a large set of instances derived from benchmarks from the literature.

Keywords: knapsack problem, mathematical models, dynamic programming, computational experiments.

1 Introduction

Integer Programming and combinatorial optimization problems require to determine the optimal values for a set of variables, each having a discrete domain. In many cases, variables enforce boolean conditions, and it is quite natural to resort to binary variables. Just to mention a few examples, in the knapsack problem one has to decide whether to insert an item in the knapsack or not. Similarly, in scheduling applications, one is asked to decide if a job should be scheduled on a given machine. Finally, the vehicle routing problem asks to decide if a certain customer must be included in a certain route and if a given edge has to be used in the solution or not. The explosion of new results for the solution of binary problems in the last 30 years is motivated by the amount and relevance of applications that can be tackled with these models. It turns out that in many relevant real-world problems decisions can also be taken at a fractional level and thus decision variables can attain non-integer real values. However, this additional freedom of "splitting" an integer variable and selecting only a fractional part will likely incur additional costs, i.e. "penalties" for deviating from integrality, thus worsening the solution value.

For example, in preemptive scheduling (see, Pinedo [16]), each task may be processed in different phases, until it has finished its execution, to minimize the total makespan. In the Split Delivery

Vehicle Routing Problem (see, Archetti and Speranza [1]), the restriction that each customer has to be visited exactly once is removed, i.e., each customer can be served by more than one route, possibly reducing the total cost of the solution. In most of the cases addressed in the literature, splitting an *item* either produces no additional cost or gives a constant penalty; e.g., Malaguti, Medina and Toth [13] considered a two-dimensional cutting problem in which raw material has to be cut to produce items, and each cut introduces some constant trim loss. In some applications, the deterioration of the solution induced by splitting cannot be evaluated a priori, hence some approximation has to be used; e.g., Lodi et al. [12] considered an applications arising in Mobile WiMAX in which data (items) have to be sent from a base station to users using a unique channel (the knapsack). In this system model, a part of the channel is used to allocate additional information about the packets that are sent. Splitting an item is allowed, but it increases the amount of additional information to be sent, i.e., it reduces the available capacity. As the objective is to minimize the amount of overhead while transmitting all data, the problem was formulated to minimize the number of items that are fractioned.

In this paper we make a step further in the study of integer problems in which splitting is allowed by removing the assumption that the penalty induced by splitting is a constant[1]. In particular, we allow the penalty to be described by an arbitrary function that depends on the fraction of item that is taken, and apply this setting to the simplest combinatorial optimization problem, namely to the 0-1 Knapsack Problem (KP) (cf. [15],[11]).

In KP we are given a knapsack of capacity W and a set $N = \{1, \dots, n\}$ of items, each item $j \in N$ having a positive weight $w_j \le W$ and a positive profit p_j. The problem asks for selecting a subset of items with maximum profit whose total weight does not exceed the knapsack capacity. As items cannot be split, KP can be modeled by associating, to each item j, a binary variable x_j taking value 1 if the item is selected, and 0 otherwise. Hence, the profit for each item is expressed as $p_j x_j$, and the capacity it consumes is $w_j x_j$. In the *Fractional Knapsack Problem with Penalties* (FKPP) addressed in this paper, fractions of items are allowed to be selected, but whenever an item is split, a penalty is incurred. Thus, the net profit associated with a fraction $0 < x_j < 1$ of an item j is smaller than (or equal to) $p_j x_j$, while no profit is earned when the item is not selected, and the full item profit p_j is earned for $x_j = 1$. Formally, FKPP is defined as follows: Given a knapsack problem KP as defined above, for each item j there is a function $F_j : [0,1] \to \Re$ such that $F_j(x_j)$ represents the profit earned if item j is taken at some (possibly, fractional) value $x_j \in [0,1]$. We assume that each function $F_j(\cdot)$ has the following shape:

$$F_j(x_j) = \begin{cases} 0 & \text{if } x_j = 0 \\ p_j & \text{if } x_j = 1 \qquad (j \in N) \\ p_j x_j - f_j(x_j) & \text{otherwise} \end{cases} \tag{1}$$

where $f_j(x_j) \ge 0$ for $x_j \in [0,1]$ is an arbitrary (even discontinuous) function representing the penalty incurred in case item j is taken at a fractional level x_j. Observe that we allow $f_j(0) > 0$ and/or $f_j(1) > 0$ for some j, as these values are immaterial for the definition of function $F_j(\cdot)$. In the general case we will not impose any further restrictions on $f_j(\cdot)$ except that function values can be computed in constant time. Thus, FKPP can be formulated as

$$\max \left\{ \sum_{j=1}^{n} F_j(x_j) : \sum_{j=1}^{n} w_j x_j \le W, \quad 0 \le x_j \le 1 \quad (j \in N) \right\}$$

where each item j has associated a continuous variable x_j indicating the fraction of item j that is selected, and functions $F_j(\cdot)$ are defined according to (1).

[1]Other penalty functions appeared as auxiliary subproblems in [10] and [5].

A special case of FKPP is given by the *Continuous Knapsack Problem* (CKP), that is the relaxation of KP obtained removing the integrality requirement of the variables. In this case, variables x_j have the same meaning as in FKPP, and both the earned profit and the used capacity are proportional to x_j. Thus, CKP arises when $f_j(x_j) = 0 \ \forall x_j \in [0, 1]$ and for each item j. It is well known that this relaxation of KP can be solved in polynomial time by ordering items according to a non decreasing profit over weight ratio, and inserting them into the knapsack in this order. The first item that cannot be completely inserted in the knapsack (if any), the so-called *critical element* (also known as split or break item), is fractionally inserted in the knapsack, so as to saturate its capacity, and the corresponding fraction of profit is earned.

Literature review. The classic KP is weakly NP-hard, and in practice fairly large instances can be solved to optimality with moderate computational effort. The reader is referred to the books by Martello and Toth [15] and by Kellerer, Pferschy and Pisinger [11] for comprehensive discussion on algorithms, applications and variants of the problem. Despite the wide existing literature on knapsack problems, only few contributions can be found that explicitly take penalties into account.

Freling, Romeijn, Morales and Wagelmans [10] considered a reformulation of the Multiperiod Single-Sourcing Problem as a Generalized Assignment Problem and noticed that the pricing problem is a knapsack problem with penalties. In this problem, that they called Penalized Knapsack Problem, the objective function includes a penalty that is described by a convex function depending on the total amount of capacity that is used. Observing that the resulting objective function is concave, the authors analyzed the structure of an optimal solution to the continuous relaxation of the problem. They conclude that it has the same structure of and optimal solution to the CKP, and propose a solution algorithm. Ceselli and Righini [7] considered another version of the Penalized Knapsack Problem in which each item has associated a constant penalty and the objective function is given by the total profit minus the largest penalty among the selected items. This problem was extensively studied in Della Croce et al. [8].

Another related problem is the Bin Packing Problem with Item Fragmentation (BPPIF), that was introduced by Mandal, Chakrabarti and Ghose [14] to model an application arising in VLSI circuit design. In this problem, one is asked to pack a given set of items in a fixed number of identical bins, while minimizing the number of items that are split among different bins. Casazza and Ceselli [5] formulated BPPIF using a mathematical model with an exponential number of variables, requiring the definition of column generation techniques. It turns out that the pricing problem in this formulation is a special case of FKPP, where the penalty for selecting each item j at a fractional level x_j is defined as a linear function $f_j(x_j) = k_j(1 - x_j)$ for each $j \in N$ and $x_j \in (0, 1)$. Casazza and Ceselli [6] introduced mathematical models and algorithms for many variants of BPPIF. Recently, Byholm and Porres [4] noticed that BPPIF arises in the operation of file systems, and presented approximation and metaheuristic algorithms for its solution. As mentioned, in all these papers there is a constant penalty associated with the splitting of an item for bin packing.

FKPP is also related to the general nonlinear knapsack problem. Bretthauer and Shetty [3] presented a survey concerning algorithms and applications of this problem, and analyzed the general form of the problem and of its most common variants: continuous or integer variables, convex or nonconvex functions, separable or nonseparable functions, bounds on the variables, or generalized upper bound (GUB) constraints. None of these variants, however, can be used to model FKPP.

Paper contribution. To the best of our knowledge, this is the first paper that specifically addresses FKPP in its general settings. Our contributions can be summarized as follows:

1. General structural properties, the special case where all profit functions F_j are convex, and the

analogous case where penalties occur as additional weights (instead of costs) are introduced in Section 2.

2. In Section 3 we propose two different mathematical models and discuss the relation between the two models. The first model has a linear number of variables and constraints, but a non-linear objective function. The second model restricts the weight contribution of each variable to integers and resembles a Multiple-Choice Knapsack Problem (MCKP), albeit of pseudopolynomial size. Moreover, we construct a Fully Polynomial Time Approximation Scheme (FPTAS) for the problem in its general form. This differs from the classical approaches since a special treatment of the profit space by means of inverse profit functions is required.

3. From an algorithmic point of view we first report in Section 4 the dynamic program recently proposed by Ceselli and Casazza [5] for the optimal solution of a special case of the problem. Then an improved algorithm with lower computational complexity is presented which partitions the dynamic programming iterations into phases of suitable size. Finally, Section 5 presents some fast and simple heuristic algorithms for the approximate solution of FKPP.

4. Section 6 reports the outcome of an extensive computational study on the performance of the proposed models and algorithms. To this end we developed benchmark instances derived from the KP literature using different shapes of the penalty functions. It turns out that our newly developed, improved dynamic programming scheme delivers the best performance among all solution approaches for the instances it applies to.

2 Structural results

In this section we impose some natural assumptions on the input data, and describe some properties of any optimal FKPP solution that will be used in the next section for modelling and solving the problem. Furthermore, we will introduce the relevant special case where all $F_j(\cdot)$ are convex.

Assumption 1 *All item weights w_j and the capacity value W are integers.*

This assumption is without loss of generality, as fractions, if any, can be handled by multiplying with a suitable factor.

Proposition 1 *For each item $j \in N$ we can replace $F_j(\cdot)$ with $\tilde{F}_j(\cdot)$ in the objective function of FKPP, where $\tilde{F}_j(x) = \max_{y \leq x_j}\{F_j(y)\} \, \forall x_j \in [0,1]$.*

Proof. Let x^* be an optimal FKPP solution. By contradiction, let j be an item such that $F(x_j^*) < F(y)$ for some $y < x_j^*$. Reducing the value of x_j^* to y produces a solution that satisfies the capacity constraint and has a higher profit; thus x^* cannot be an optimal solution. $\square$

Based on this proposition, our second assumption follows:

Assumption 2 *For each item $j \in N$, function $F_j(\cdot)$ is non-decreasing in $[0,1]$.*

We now observe that, while an optimal solution to CKP includes at most one fractional item (namely, the critical item), this is not the case for FKPP. Indeed, there are generic FKPP instances with n items for which the optimal solution splits *all* items.

Proposition 2 *There are instances of FKPP for arbitrary n, where all n items are split in the optimal solution.*

Proof. Consider the following instance with n items, knapsack capacity $W = M$ for some large value M, and all items identical with $p_j = w_j = M-1$ for $j = 1, \dots, n$. The profit function $F_j(\cdot) = F(\cdot) = (M-1)x - f(x)$ for $j = 1, \dots, n$ is defined as the following piece-wise linear (non-convex) function (see Figure 1):

$$F(x_j) = \begin{cases} 0 & \text{if } 0 \le x_j \le \frac{2}{M-1} \\ \frac{M}{M-2n}(M-1)x_j - \frac{2M}{M-2n} & \text{if } \frac{2}{M-1} < x_j \le \frac{1}{n}\frac{M}{M-1} \\ \frac{M}{n} & \text{if } \frac{1}{n}\frac{M}{M-1} < x_j \le \frac{M-2}{M-1} \\ (M-1)(M-1-\frac{M}{n})x_j - (M-1)(M-2-\frac{M}{n}) & \text{if } \frac{M-2}{M-1} < x_j \le 1 \end{cases}$$

Choosing an item j with $x_j \ge \frac{M-2}{M-1}$ leaves a residual capacity ≤ 2 which could only be filled by an item i with $x_i \le \frac{2}{M-1}$, but then item i would contribute zero profit and the resulting solution has a total profit at most $p_j x_j \le M-1$.

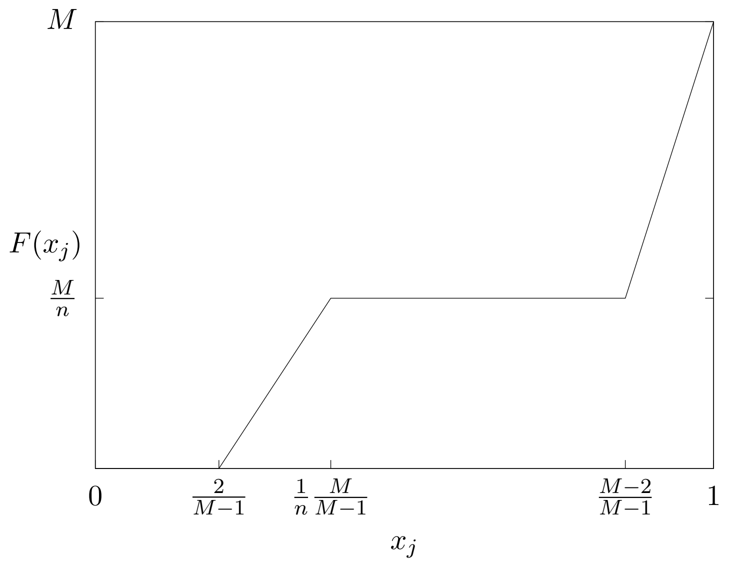

Figure 1: Example of a generic profit function that leads to an optimal solution with all fractional items.

For choosing an item j with $x_j < \frac{M-2}{M-1}$ it is always better to set x_j to the lower end of the interval where profit is constant, i.e. $x_j = \frac{1}{n}\frac{M}{M-1}$. Taking this fractional choice for all n items yields an optimal solution value of M (and total weight M). Choosing a value even smaller than $\frac{1}{n}\frac{M}{M-1}$ for some items does not offer any opportunities for improving the contribution of other items by increasing their values, since an increase of profit starts only for item values $> \frac{M-2}{M-1}$, which leads us back to the case we settled at the beginning of the proof. $\Box$

2.1 FKPP with convex objective function

For several applications, it is reasonable to assume that the penalty functions $f_j(\cdot)$ are concave. This means that taking a smaller part of an item and selecting it (or its complement) will incur a rather small penalty, as it may correspond to a minor trimming operation. However, splitting an item closer to the middle, and thus deviating more significantly from integrality, requires a more involved effort and and causes a higher penalty. Hence, we will consider the special case where all functions $f_j(\cdot)$ are *concave* and thus all profit functions $F_j(\cdot)$ are *convex*. Clearly, this case also includes the most plausible cases of linear and constant penalty functions. Note that the special case of FKPP resulting from BPPIF (Ceselli and Casazza [5]) contains linear penalty functions and thus falls into this category of convex profit functions.

We now show that for this case an optimal solution exists in which at most one item is taken at a fractional level. This fact has pleasing modeling and algorithmic consequences, discussed next.

Theorem 1 *If all profit functions $F_j(\cdot)$ are convex, there exists an optimal solution for FKPP that has at most one fractional item.*

Proof. Let x^* be an optimal solution for FKPP that includes two items, say h and k, at a fractional level. We will show that x^* cannot be unique. Let $\varepsilon > 0$ be a positive value and define $\varepsilon_h = \varepsilon / w_h$ and $\varepsilon_k = \varepsilon / w_k$. Now consider two solutions y and z as follows:

$$y_j = \begin{cases} x_j^* & \text{if } j \neq h, k \\ x_h^* + \varepsilon_h & \text{if } j = h \\ x_k^* - \varepsilon_k & \text{if } j = k \end{cases} \qquad (j \in N)$$

and

$$z_j = \begin{cases} x_j^* & \text{if } j \neq h, k \\ x_h^* - \varepsilon_h & \text{if } j = h \\ x_k^* + \varepsilon_k & \text{if } j = k \end{cases} \qquad (j \in N)$$

Given their definition, these solutions satisfy the capacity constraint, as ε units of capacity are moved from h to k or vice-versa. In addition, as both x_h^* and x_k^* are strictly positive and smaller than 1, solutions y and z are feasible if ε is chosen such that $\varepsilon_h \leq \min\{1 - x_h^*, x_h^*\}$ and $\varepsilon_k \leq \min\{1 - x_k^*, x_k^*\}$. Finally, $\varepsilon > 0$ implies $\varepsilon_h > 0$ and $\varepsilon_k > 0$, i.e., y and z are distinct from x^* and $x^* = \frac{1}{2}y + \frac{1}{2}z$. As all profit functions are convex we have that:

$$\sum_{j \in N} F_j(x_j^*) \leq \frac{1}{2} \sum_{j \in N} F_j(y_j) + \frac{1}{2} \sum_{j \in N} F_j(z_j)$$

implying that:

$$\sum_{j \in N} F_j(x_j^*) \leq \max \left\{ \sum_{j \in N} F_j(y_j), \sum_{j \in N} F_j(z_j) \right\}$$

i.e., at least one between y and z yields a profit not smaller than that of x^*. Thus, another optimal solution may be defined either increasing h and decreasing k (or vice-versa), until one of the two variables hits either the lower or the upper bound. □

Since we assumed the knapsack capacity W and all the item weights w_j ($j = 1, \ldots, n$) to be integer, the following result is a direct consequence of Theorem 1.

Lemma 1 *If all profit functions $F_j(\cdot)$ are convex, there exists an optimal solution to FKPP where $w_j x_j$ is integer for each item j.*

Proof. Theorem 1 ensures that an optimal FKPP solution, say x^*, exists in which only one item is split. Let k be such item. Since we assume all weights and the capacity be integer, all items $j \neq k$ have an integer value of $w_j x_j^*$, and the residual capacity for item k is an integer number. As function $F_k(\cdot)$ is non-decreasing, there is an optimal solution where item k is taken at the largest possible value, which is indeed an integer. $\square$

2.2 Penalty in terms of weight

We have defined FKPP by considering a penalty in the profit of fractional items. It seems natural to also consider the case in which the penalty for fractional items is enforced in terms of additional weight, instead of reduced profit. In other words, we can consider that a fraction x_j of a given item j produces a profit $p_j x_j$, but consumes a capacity $G_j(x_j) \geq w_j x_j$, where $G_j(x_j)$ is a (possibly discontinuous) function defining the weight of fraction x_j, once the penalty is considered. Clearly, $G_j(0) = 0$ and $G_j(1) = w_j$, for all $j \in N$. In this section we show that the fractional knapsack problem with weight penalty can be reduced to FKPP for a suitable penalty function.

Proposition 3 *Without loss of generality, for all $j \in N$ we can define $G_j(.)$ only for those values of $x_j \in [0, 1]$ for which $G_j(.)$ is a* strictly *increasing function.*

Proof. Let x^* be an optimal solution of the fractional knapsack problem with weight penalty. By contradiction, let j be an item such that $G_j(x_j^*) \geq G_j(y)$ for some $y > x_j^*$. Increasing the value of x_j^* to y produces a solution that satisfies the capacity constraint (the actual capacity consumption for item j is not increased) and has a higher profit; thus x^* cannot be an optimal solution. $\square$

Based on this proposition, we can assume:

Assumption 3 *For all $j \in N$, function $G_j(.)$ is a* strictly *increasing function and hence it is invertible.*

Note that, if $G(.)$ is discontinuous then $G^{-1}(.)$ is not defined for some weight values. However, we can complete the definition of each $G_j^{-1}(.)$ for the whole domain by setting $G_j^{-1}(w) := \max\{x_j : G_j(x_j) \leq w\}$ for all $w \in [0, w_j]$.

The main result of this section follows:

Proposition 4 *A fractional knapsack problem with penalty in terms of weight, described by invertible functions $G_j(.)$ for all $j \in N$, can be reduced to a FKPP.*

Proof. For each $j \in N$, let $\gamma_j = \lim_{x_j \to 0^+} G_j(x_j)$. The problem with weight penalty is equivalent to FKPP where, for each $j \in N$, $F_j(x_j)$ is defined as follows:

$$F_j(x_j) = \begin{cases} 0 & \text{if } x_j < \gamma_j / w_j \\ \frac{p_j}{w_j} G_j^{-1}(w_j x_j) & \text{if } x_j \geq \gamma_j / w_j \end{cases} \tag{2}$$

where $G_j^{-1}(.)$ is the completed inverse function of $G_j(.)$. Thus, the profit function is zero for small values of x_j, while for $x_j \geq \gamma_j / w_j$, $G_j(x_j) \geq w_j x_j$ implies $F_j(x_j) \leq p_j x_j$, i.e., the profit is decreased by some penalty with respect to the linear case. $\square$

3 Mathematical models for FKPP

In this section we present two alternative mathematical models for FKPP. The first model is general, though it may be non-linear depending on the form of the penalty functions. Conversely, the second model is an Integer Linear Program (ILP) restricted to integral weight contributions of all items. Recalling Lemma 1, this applies e.g. for the case when all $F_j(\cdot)$ functions are convex.

3.1 General model

The first model ($MGEN$) has continuous variables x_j to denote the fraction of item $j \in N$ in the solution. In addition we introduce, for each item j, two binary variables α_j and β_j, that are used to handle the profit function for $x_j = 0$ and $x_j = 1$, respectively. Denoting by $\Delta_j^0 = f_j(0)$ and $\Delta_j^1 = f_j(1)$, the formulation reads:

$$(MGEN) \qquad \max \sum_{j \in N} \left(p_j\, x_j - f_j(x_j) + \Delta_j^0\, \alpha_j + \Delta_j^1\, \beta_j \right) \tag{3}$$

$$\sum_{j \in N} w_j\, x_j \le W \tag{4}$$

$$\alpha_j \le 1 - x_j \qquad j \in N \tag{5}$$

$$\beta_j \le x_j \qquad j \in N \tag{6}$$

$$0 \le x_j \le 1 \qquad j \in N \tag{7}$$

$$\alpha_j, \beta_j \in \{0, 1\} \qquad j \in N. \tag{8}$$

The objective function (3) takes into account both the linear profit and the penalty of each item and is possibly increased by Δ_j^0 or Δ_j^1 when item j is not selected or fully inserted into the knapsack, respectively. While (4) is the capacity constraint, inequalities (5) and (6) set the correct values of the α_j and β_j variables, allowing each such variable to be 1 only in case the associated item is not taken or fully taken, respectively.

This model has polynomial size for what concerns both the number of variables and constraints. All constraints in the model are linear, and all possible nonlinearities appear in the objective function only. Thus, the computational tractability of the model depends on the shape of the penalty functions $f_j(\cdot)$.

3.2 An ILP model for integer weight contributions

The second model ($MINT$) assumes the integrality of the used weight for each item j. It samples function $F_j(x_j)$ for all relevant values of variable x_j, and introduces a binary variable for each such value. In particular, for each item $j \in N$ and possible weight $k = 1, \ldots, w_j$, we introduce a binary variable y_{jk} that takes value 1 if item j uses k units of capacity, i.e., if $w_j x_j = k$. For each pair (j, k), we denote by t_{jk} the net profit obtained by taking $\frac{k}{w_j}$ units of item j, namely $t_{jk} = F_j(\frac{k}{w_j}) = p_j \frac{k}{w_j} - f_j(\frac{k}{w_j})$.

Recall that, for the case where all profit functions $F_j(\cdot)$ are convex, Lemma 1 ensures that an optimal solution exists in which each item uses an integer number of units of capacity. Thus, $MINT$ gives an optimal solution for the convex case. Of course, $MINT$ can be also applied for the general case where it will yield a possibly sub-optimal, approximate solution, which will be discussed in Section 3.3.

Using the above definitions, the second model is:

$$(MINT) \qquad \max \sum_{j \in N} \sum_{k=1}^{w_j} t_{jk}\, y_{jk} \tag{9}$$

$$\sum_{k=1}^{w_j} y_{jk} \leq 1 \qquad j \in N \tag{10}$$

$$\sum_{j \in N} \sum_{k=1}^{w_j} k\, y_{jk} \leq W \tag{11}$$

$$y_{jk} \in \{0, 1\} \qquad j \in N; k = 1, \ldots, w_j. \tag{12}$$

Objective function (9) takes into account both the profit and the penalties for the selected items. Constraints (10) ensure that each item is associated with a unique weight value, whereas (11) imposes the capacity constraint.

Observe that the number of y_{jk} variables is $\sum_{j \in N} w_j$, i.e., pseudopolynomial in the input size. Thus, we may expect this model to be extremely challenging to be solved for instances with large item weights. However, this model is a pure Integer Linear Program (ILP), and can thus be solved using the rich machinery offered by modern commercial ILP solvers. Moreover, it is easy to see that $MINT$ corresponds to a *Multiple-Choice Knapsack Problem* (MCKP) with inequality constraints, where each subset of items in MCKP corresponds to the possible choices for cutting an integer weight from an item in FKPP. Thus, one could also solve $MINT$ by applying specialized algorithms for MCKP. Note that number of items in the resulting instance of MCKP, say $\widetilde{n}$, is pseudopolynomial, namely $\widetilde{n} = \sum_{j \in N} w_j$ which is in $O(nw_{\max})$, where $w_{\max} := \max\{w_j : j = 1, \ldots, n\}$ denotes the maximum weight of an item. Thus, the classical dynamic programming scheme described in [11, Sec. 11.5]) would take $O(nw_{\max}W)$ time.

3.3 Comparison between $MGEN$ and $MINT$

In the convex case both the general model $MGEN$ and model $MINT$ are valid. The following proposition states the relationship of the respective continuous relaxations.

Proposition 5 *When functions $F_j(\cdot)$ are convex for all $j \in N$, the continuous relaxation of model* (3)–(8) *dominates the continuous relaxation of model* (9)–(12)*.*

Proof. Let (x^*, α^*, β^*) be an optimal solution to the continuous relaxation of model $MGEN$. Observe that, in any optimal solution, variables α and β will be at the maximum value that is allowed, namely $\alpha_j^* = 1 - x_j^*$ and $\beta_j^* = x_j^*$ for each $j \in N$. Thus, the contribution of each item j to the objective function is

$$p_j x_j^* - f_j(x_j^*) + \Delta_j^0(1 - x_j^*) + \Delta_j^1 x_j^* \leq$$
$$p_j x_j^* - \left(\Delta_j^0(1 - x_j^*) + \Delta_j^1 x_j^*\right) + \Delta_j^0(1 - x_j^*) + \Delta_j^1 x_j^* = p_j x_j^*$$

where the first inequality holds by convexity of $-f(x_j)$.

Consider now a solution, say y^*, for the second model, defined as follows: for each item $j \in N$, let

$$y_{jk}^* = \begin{cases} 0 & \text{if } k < w_j \\ x_j^* & \text{if } k = w_j \end{cases}$$

It is easy to see that this solution satisfies constraints (10) and (11), i.e., y^* is a feasible solution to the continuous relaxation of $MINT$. The associated value is $p_j y_{jw_j}^* = p_j x_j^*$ which concludes the proof. □

We can also explicitly state the following characterization of the continuous relaxation of $MINT$.

Proposition 6 *Let $\bar{y}_{jk}$ be an optimal solution of the LP-relaxation of $MINT$. Then $\bar{y}_{jk} = 0$ for $k < w_j$.*

Proof. Comparing the efficiencies e_{jk} (profit per unit of weight) we have for $k < w_j$:

$$e_{jk} = \frac{p_j \frac{k}{w_j} - f_j(\frac{k}{w_j})}{k} = \frac{p_j}{w_j} - \frac{1}{k} f_j(\frac{k}{w_j}) \leq \frac{p_j}{w_j} = e_{jw_j}$$

Considering (10), it is always advantageous to concentrate the solution values assigned to the variables corresponding to item j to the variable y_{jw_j} with largest efficiency. □

Although the associated continuous relaxation of $MINT$ provides weaker bounds than its counterpart for $MGEN$, the former can be computed much more efficiently. It follows from Proposition 6 that the LP-relaxation of $MINT$ is equivalent to the LP-relaxation of the underlying knapsack problem KP and thus can be solved by the classical linear time algorithm (see [11, Sec. 3.1]) in $O(n)$ time[2]. For the continuous relaxation of $MGEN$ no combinatorial algorithm is available for the general case.

As observed above, one could obtain an approximation of the general case by solving model $MINT$ and thus implicitly adding the restrictions that $w_j x_j$ is integer for each item j. This might seem attractive since the MCKP-type model $MINT$ can be expected to be much easier to solve than the general $MINT$, as illustrated by our experiments in Section 6. Unfortunately, the quality of this approximation may be quite low in the worst case.

For a given instance I, let us denote by $z^*(I)$ and $z_{INT}(I)$ the optimal solution value and the value of the solution computed by model $MINT$, respectively. The following result determines the maximum percentage deviation of the latter with respect to the optimum when nonconvex profit functions are considered.

Theorem 2 *Let I be an instance of FKPP with arbitrary profit functions $F_j(\cdot)$ for each item j. Then, we have $\frac{z_{INT}(I)}{z^*(I)} \leq \frac{1}{2}$ and the ratio can be tight.*

Proof. For ease of notation, we omit the indication of the instance I at hand. Let z^C be the optimal solution value of the problem in which items can be split with no penalties. Similarly, denote by z^{KP} the same value for the problem in which they cannot be split at all. We have

$$z^C \geq z^* \geq z_{INT} \geq z^{KP} \geq \frac{z^C}{2}, \tag{13}$$

where the last inequality derives from the observation that z^{KP} corresponds to a KP whose continuous relaxation is z^C. Thus, both z_{INT} and z^* belong to the interval $[\frac{z^C}{2}, z^C]$, implying that the ratio between these two values cannot be larger than $\frac{1}{2}$.
To prove that the ratio is tight, consider a family of instances defined according to an integer parameter k, as follows: there are $n = k^2 + 1$ items and the capacity is $C = k + 1$. The first item

[2]Note that employing the linear time algorithm for the LP-relaxation of MCKP (see [11, Sec. 11.2]) would give an $O(nw_{\max})$ pseudopolynomial algorithm.

has profit $p_1 = k$, weight $w_1 = 1$, and a profit function such that $F_1(x_1) = 0$ for $x_1 < 1$. Each remaining item $j = 2, \dots, n$ has $p_j = k+1$, $w_j = k+1$, and profit function $F_j(x_j) = (k+1)x_j$ for $0 \le x_j \le \frac{1}{k(k+1)}$ and $F_j(x_j) = \frac{1}{k}$ for $x_j \ge \frac{1}{k(k+1)}$.
The optimal solution completely packs the first item and fills the residual capacity by taking each remaining item j at a level $x_j = \frac{1}{k(k+1)}$. It is easy to see that each such item requires a capacity equal to $\frac{1}{k}$ and that the resulting profit is $z^* = p_1 + k^2\frac{1}{k} = k + k = 2k$. The heuristic solution is composed by a unique item $j \in [2, n]$ that is taken completely, thus yielding a profit $z_{INT} = k+1$. Observe that an equivalent solution is obtained taking item 1 and using the residual capacity equal to k to pack the remaining items. In this case, forcing each item to have an integer $w_j x_j$ value implies that k items are taken at a level $x_j = \frac{1}{k+1}$, thus producing a total profit equal to $k + k\,\frac{1}{k} = k+1$. Thus we have $\frac{z_{INT}}{z^*} = \frac{k+1}{2k}$, i.e, the ratio is arbitrarily close to $\frac{1}{2}$ for sufficiently large k. □

Inequality (13) also shows the following bound on the effect of allowing fractional values.

Corollary 1

$$z^{KP} \le z^* \le 2z^{KP}$$

3.4 An FPTAS for the general case

We can derive a Fully Polynomial Time Approximation Scheme (FPTAS) for FKPP by employing another approximate representation of FKPP as a Multiple-Choice Knapsack Problem (MCKP) different from Section 3.2.

For each item j, we define a function $\varphi_j : [0, p_j] \to [0, 1]$ such that $\varphi_j(p) = \min\{x : F_j(x) \ge p\}$. This function is well defined since all F_j are monotonically increasing, though it may not be easy to evaluate. If F_j is strictly increasing and continuous then φ_j is the inverse function F_j^{-1}. According to our definition, if a profit function is constant on a certain interval, i.e. $F_j(x) = c$ for all $x \in [a, b] \subseteq [0, 1]$, then $\varphi_j(c) = a$.

Note that in all cases we are aware of, an FPTAS for knapsack-type problems is derived from an exact, pseudopolynomial dynamic programming scheme. Scaling the profit space then leads to a polynomial running time at a loss of optimality. For FKPP the situation is different since the continuous, non-discrete nature of the decision variables and the resulting item weights and profits does not seem to permit an exact dynamic programming approach for the general case. Of course, it is well-known that the MCKP instance implied by $MINT$ can be solved to optimality in pseudopolynomial time and can be approximated by an FPTAS, but the resulting solution may deviate considerably from the optimal solution for the general case of FKPP, as illustrated by Theorem 2.

For our general FPTAS we are given an accuracy parameter ε and a scaling parameter K to be defined later. We partition the profit range $[0, p_j]$ of every item j into equal-sized intervals of width K. Thus, the profit range is approximated by a sequence of discrete values $0, K, 2K, \dots, \lfloor \frac{p_j}{K} \rfloor K$, with a smaller interval possibly left just before p_j. This guarantees that for any profit $p \in [0, p_j]$, there exists an integer number $i \in \{0, 1, \dots, \lfloor \frac{p_j}{K} \rfloor\}$ such that $iK \le p < (i+1)K$.

For any instance I of FKPP, we define a corresponding instance I^A of MCKP with the same capacity W. Each item j in I gives rise to a subset N_j of $\lfloor \frac{p_j}{K} \rfloor + 1$ items in I^A. For $i \in \{0, 1, \dots, \lfloor \frac{p_j}{K} \rfloor\}$ there is an item in N_j with profit i and weight $\varphi_j(i\,K)$. Note that the sequence of profits in each subset is completely regular, while the sequence of weights may contain large jumps and also subsequences of identical values if F_j is discontinuous. Every feasible solution of I^A directly implies a feasible solution of I by setting $x_j = \varphi_j(i\,K)$ if item i was selected from subset N_j.

Our FPTAS consists of running dynamic programming by profits to solve I^A to optimality and reporting the associated feasible solution for I. This can be done by a standard algorithm (see [11, Sec. 11.5]) whose running time is a product of the number of items, and an upper bound on the objective function. Setting $p_{\max} := \max\{p_j : j = 1, \dots, n\}$, considering that the total number of items in I^A is $\sum_{j \in N}(\lfloor \frac{p_j}{K} \rfloor + 1) \leq n\, p_{\max}/K + n$, and stating a trivial upper bound on the objective function value of I^A as $n\, p_{\max}/K$, its running time can be bounded by $O((np_{\max}/K)^2)$. Now, choosing (similar to the classical FPTAS for KP) $K := \frac{\varepsilon p_{\max}}{n}$ the running time is fully polynomial, namely $O(n^4/\varepsilon^2)$. Note that we do not consider possible improvements of this complexity but just want to establish the existence of an FPTAS.

Theorem 3 *There is an FPTAS for FPKK if values $\varphi_j(p)$ can be computed in polynomial time for all $j \in N$ and for each p.*

Proof. It remains to show that the algorithm yields an ε-approximation.[3] Consider an instance I with optimal solution x^*, solution value z^*, and the corresponding instance I^A of MCKP. For each value x^*_j, determine an integer number i_j such that $i_j K \leq F_j(x^*_j) < (i_j + 1)K$. Now define the following solution, obtained by setting $x^{*A}_j := \varphi_j(i_j\, K)$ for each item j. Clearly, x^{*A} is a feasible solution in I^A; let us denote by z^{*A} its objective function value in I^A. Observe that x^{*A} is also feasible in I, and its associated value is Kz^{*A}. The difference between z^* and Kz^{*A}, i.e. moving from I to I^A and back to I, can be bounded by

$$z^* - Kz^{*A} \leq nK = \varepsilon p_{\max} \leq \varepsilon z^* \tag{14}$$

since the profit contributed by each item j will be reduced by less than K.

Now consider the solution to I^A computed by our algorithm, i.e., an optimal solution obtained running the dynamic programming algorithm, and let $z^A \geq z^{*A}$ be its objective value in I^A. As the outcome of our FPTAS, this implies a feasible solution, say x', for I with objective function value $z' = Kz^A$. Thus, we have from (14)

$$(1 - \varepsilon)z^* \leq Kz^{*A} \leq Kz^A = z'$$

i.e., the produced solution yields an ε-approximation of z^* for any $\varepsilon > 0$. □

4 Dynamic Programming algorithms for the convex case

In the following sections, we present dynamic programming (DP) algorithms that can be used for computing an optimal solution to FKPP in case that an optimal solution exists with at most one fractional item. By Theorem 1 this applies for the broad class of instances where profit functions are convex. Our algorithms will use a DP algorithm for KP as a black box. Thus, we firstly review two classical DP algorithms for KP. Then, we describe the DP approach proposed in [5] to solve FKPP. Finally, we present a new algorithm that yields improvements with respect to the algorithm by [5] both from a theoretical and from a computational viewpoint.

4.1 Dynamic Programming algorithms for KP

The basic idea of DP algorithms for KP is to solve, for each item $j = 1, \dots, n$ and capacity value $c = 1, \dots, W$, the KP instance defined by item set $\{1, \dots, j\}$ and capacity c. Denoting by $T(j, c)$ the value of this solution, we can determine an optimal solution to the original instance as $T(n, W)$.

[3] It should be noted that the direct application of standard arguments as given in [11, Sec. 2.6] does not work since the optimal solution for I may have an almost arbitrary profit if translated into a feasible solution of I^A.

There are two main procedures that can be used to compute the T entries. The first one, also known as Bellman recursion [2], is based on the computation of *all* the $n \times W$ entries. The computation of each entry can be done in constant time, yielding an $O(nW)$ complexity. The second scheme is known as *Dynamic Programming with Lists* (see, [11, ch. 3.4]). The basic idea is that an optimal solution can be determined without computing all the $T(j, c)$ entries since many (j, c) pairs may be redundant. This consideration may reduce the memory requirement and improve the computational performance of the resulting algorithm. On the other hand, it has no effect in reducing the worst-case complexity and, if the number of non-redundant entries is comparable to nW, then the overhead needed to handle the lists could be significant.

In the following, we will assume that a black box procedure `DP-KP` implementing one of the schemes above is available. Observe that both algorithms require some initialization step; conceptually, this corresponds to setting $T(0, c) = 0$ for each $c = 1, \ldots, W$. However, one can easily generalize the schemes simply setting $T(0, c) = S(c)$ for each $c = 1, \ldots, W$, where S is some starting vector to be properly defined in case some profit is obtained even when no items are taken ($S = \underline{0}$ means that a zero vector is used). In addition, note that the DP algorithm returns, as a byproduct, the optimal solution value for *all* possible capacities $c = 1, \ldots, W$. Given an item set N, a capacity value W, and a vector S containing a starting profit value for each capacity entry, procedure `DP-KP`(N, W, S) computes and returns a vector $\overline{T}$ such that $\overline{T}(c) = T(|N|, c)$ for each $c = 1, \ldots, W$.

4.2 A Dynamic Programming algorithm for FKPP

The DP algorithm proposed in [5] is based on the following observation: if no item is fractionally selected in an optimal solution, then an optimal solution of FKPP corresponds to the optimal solution of KP. Otherwise, one can exploit the fact that only one item is taken at a fractional level, guess this item and solve a KP associated with the remaining items. In case a DP algorithm is used to solve the KP instance, one can easily compute a posteriori the amount of capacity to be used for the split item and derive an optimal FKPP solution. The complete algorithm $DP1$ is given in Algorithm 1, where each z_j denotes the profit of the optimal solution in case item j is split. The version of this algorithm, where DP with Lists is used, will be referred by $DP2$.

Algorithm $DP1$ can be executed in $O(n^2W)$ time. Indeed, computing the optimal solution when no item is split requires the execution of a dynamic programming for KP. As to the case in which an item is split, there are n iterations. At each iteration j, one has to run the dynamic programming for KP with item set $N \setminus \{j\}$ and capacity W, which requires $O(nW)$; given that, the associated z_j value can be computed in $O(w_j)$ time. Thus, the overall complexity of $DP1$ is $O(n^2W)$. The same considerations apply for $DP2$.

The following observation is immediate from the previous result:

Observation 1 *If all profit functions $F_j(\cdot)$ are convex, FKPP is weakly NP-hard.*

As observed, the algorithm above solves the problem to optimality with the required complexity. In addition, FKPP cannot be easier than KP, which arises as a special case when no item j can be split, e.g., $f_j(x_j) = p_j \quad \forall x_j \in (0, 1) \quad (j \in N)$.

4.3 An improved Dynamic Programming algorithm for FKPP

In this section we introduce a new DP algorithm for FKKP having an improved computational complexity with respect to the scheme given in the previous section. The resulting algorithm

Algorithm 1 $DP1$

```
{compute the optimal solution when no item is split}
set $\overline{T} :=$ DP-KP$(N, W, \underline{0})$ and $z^* := \overline{T}(W)$
for all items $j \in N$ do
  {apply DP without item $j$}
  set $\overline{T} :=$ DP-KP$(N \setminus \{j\}, W, \underline{0})$ and $z_j := \overline{T}(W)$;
  {complete the solution by splitting item $j$ in the best possible way}
  for $c = 1$ to $w_j - 1$ do
    if $\overline{T}(W - c) + F_j(\frac{c}{w_j}) > z_j$ then
      $z_j := \overline{T}(W - c) + F_j(\frac{c}{w_j})$
    end if
  end for
  if $z_j > z^*$ then
    $z^* := z_j$
  end if
end for
return $z^*$
```

$IDP1$ (see Algorithm 2) takes as input an integer parameter k that will be defined later. As for the previous approach, this algorithm considers one item at a time as split item, computes the optimal KP solution without this item, and completes the solution in the best possible way. The main difference is that the KP instances are solved in an incremental way. In particular, items are partitioned into (say) r subsets, each containing at most k items. For each item j, the optimal KP solution with item set $N \setminus \{j\}$ is obtained in two steps: first, the DP algorithm is used to determine an optimal KP solution without the items in the subset that includes item j. Then, this partial solution is completed with all items in the current subset except item j, using again procedure `DP-KP` with a suitable set of starting values. As stated in the following Theorem 4, this allows a reduction of the computational complexity of the algorithm. Employing an implementation of this approach based on DP with Lists will give an algorithm denoted by $IDP2$.

Theorem 4 *Algorithm $IDP1$ can be executed in $O(n^{3/2}W)$ time.*

Proof. Consider an iteration of the algorithm, associated with group (say) L_i. The first DP execution takes $O((n-k)W)$ time. Then, k executions of the DP algorithm are needed, trying all items in L_i as split item. Each execution requires $O(kW)$ time and produces a solution in which item j is not taken. Trying all possible $w_j - 1$ ways to complete this solution with item j takes $O(W)$ additional time.
Thus, the complexity of each iteration is $O\Big((nW - kW + k[kW + W]\Big)$. Executing r iterations yields a total complexity equal to $O(rnW - rkW + rk^2W + rkW)$ time.
Taking $k = \lfloor\sqrt{n}\rfloor$ and $r = \lceil n/k \rceil \approx \sqrt{n}$ we obtain the claimed complexity equal to $O(\sqrt{n}\,nW - nW + \sqrt{n}\,nW + nW) = O(\sqrt{n}\,nW)$. $\square$

Again, a similar reasoning applies for the list based $IDP2$.

Algorithm 2 *IDP1*

```
{compute the optimal solution when no item is split}
set $\overline{T} :=$ DP-KP$(N, W, \underline{0})$ and $z^* := \overline{T}(W)$
partition item set $N$ into $r$ subsets $L_i$ such that $|L_i| \leq k$ for each $i = 1, \dots, r$
for $i = 1$ to $r$ do
  {guess the set $L_i$ of items that contains the split item}
  set $\overline{T}_1 :=$ DP-KP$(N \setminus L_i, W, \underline{0})$
  for all items $j \in L_i$ do
    {apply DP to the residual items (but item $j$) starting from the optimal values associated
    with item set $N \setminus L_i$}
    set $\overline{T}_2 :=$ DP-KP$(L_i \setminus \{j\}, W, \overline{T}_1)$ and $z_j := \overline{T}_2(W)$
    {complete the solution splitting item $j$ in the best possible way}
    for $c = 1$ to $w_j - 1$ do
      if $\overline{T}_2(W - c) + F_j(\frac{c}{w_j}) > z_j$ then
        $z_j := \overline{T}_2(W - c) + F_j(\frac{c}{w_j})$
      end if
    end for
  end for
  if $z_j > z^*$ then
    $z^* := z_j$
  end if
end for
return $z^*$
```

5 Heuristics

In this section we present three simple heuristic algorithms that provide approximate solutions for the general FKPP.

5.1 First heuristic algorithm

The first heuristic ($H1$) exploits the similarity between FKPP and KP. The procedure, described in Algorithm 3, first computes an optimal solution of the KP instance obtained when items cannot be split. Then, it fills the residual capacity (if any) using a fraction of some residual item. To this aim, all items that are not packed in the KP solution are tried, and the one returning the maximum profit is selected.

Algorithm 3 *H1*

```
solve KP and let $x$ be an optimal solution
set $z^H := \sum_{j \in N} p_j x_j$ and $\bar{c} := W - \sum_{j \in N} w_j x_j$
let $j = \arg\max_{i: x_i = 0} \{F_i(\bar{c}/w_i)\}$
set $x_j := \bar{c}/w_j$ and $z^H := z^H + F_j(x_j)$
return $z^H$
```

5.2 Second heuristic algorithm

The heuristic algorithm H1 requires the solution of a KP. As this problem is NP-hard, though solvable efficiently in practice, we developed a second algorithm based on the approximate solution of the knapsack problem. In particular, we used the classical `GREEDY` procedure described in [11], that returns a KP solution that is maximal with respect to inclusion. This solution is possibly improved using a fraction of some unpacked item, as shown in Algorithm 4.

Algorithm 4 *H2*

```
execute the GREEDY algorithm for KP and let $x$ be the resulting solution
set $z^H := \sum_{j \in N} p_j x_j$ and $\bar{c} := W - \sum_{j \in N} w_j x_j$
let $j = \arg\max_{i : x_i = 0} \{F_i(\bar{c}/w_i)\}$
set $x_j := \bar{c}/w_j$ and $z^H := z^H + F_j(x_j)$
return $z^H$
```

5.3 Third heuristic algorithm

Our third heuristic produces an initial KP solution by applying a variant of the `GREEDY` procedure, called `GREEDY-SPLIT` in [11], that packs items into the knapsack until the critical item is found (and then terminates). The residual capacity (if any) is filled in an iterative way, selecting at each iteration the unpacked item that can be packed with a maximum profit. The complete algorithm is given in Algorithm 5.

Algorithm 5 *H3*

```
execute the GREEDY-SPLIT algorithm for KP and let $x$ be the resulting solution
set $z^H := \sum_{j \in N} p_j x_j$ and $\bar{c} := W - \sum_{j \in N} w_j x_j$
while $\bar{c} > 0$ do
    let $j = \arg\max_{i : x_i = 0} \{F_i(v_i) : v_i = \min(1, \bar{c}/w_i)\}$
    set $x_j := \min(1, \bar{c}/w_j)$, $\bar{c} = \bar{c} - x_j w_j$ and $z^H := z^H + F_j(x_j)$
end while
return $z^H$
```

6 Computational experiments

In this section we report the outcome of our computational experiments on FKPP. In Section 6.1 we first give some details about the implementation of the algorithm, while Section 6.2 describes our benchmark instances. Sections 6.3, 6.4 and 6.5 report the results of the exact methods for different classes of problems, while Section 6.6 reports the outcome of the experiments concerning the heuristic solution of FKPP.

6.1 Settings

All experiments were performed single thread on a computer equipped with an Intel(R) Core(TM) i7-6900K processor clocked at 3.20 GHz and 64 GB RAM under GNU/Linux Ubuntu 16.04. Each run was assigned a time limit of one hour. All DP algorithms and the heuristic algorithms were implemented in *C++*, while models *MGEN* and *MINT* were solved using the state-of-the-art commercial solver CPLEX 12.7.1.

As to model *MINT*, we also solved it using a combinatorial exact algorithm, namely algorithm *MCKP* by Pisinger [17]. This algorithm is considered the state-of-the-art in the solution of Multiple-Choice Knapsack Problems, and its code is publicly available at `www.diku.dk/~pisinger/codes.html`. Given a FKPP instance, we defined an associated MCKP instance as follows: each item j in FKPP corresponds to a subset of items of MCKP, and each possible fraction of item j with weight equal to k in FKPP corresponds to an item with weight k in subset j of the MCKP instance, possibly removing MCKP items with negative profit. Algorithm *MCKP* is designed for problems with integer positive data and for the version of the problem with the equality constraint, i.e., the case in which exactly one item must be selected from each subset of items. Thus, we had to implement the following transformation:

1. all profit values were multiplied by a large factor, possibly rounding the obtained values;
2. for each subset j of items, a dummy item with zero profit and weight was added;
3. the profit and the weight of each item were increased by one, the capacity was increased by n.

The results obtained by solving the resulting MCKP instance will be denoted by *MCKP* in the following. Computational experiments showed that *MCKP* largely outperforms, in terms of computing time, the direct application of our commercial solver on model *MINT*. For this reason, we do not report the results obtained using the latter method in our analysis.

6.2 Benchmark instances

To the best of our knowledge, there is no FKPP benchmark in the literature. Thus, we generated a large set of problems derived from KP instances from the literature. We now describe the way KP instances were selected, and discuss later how each KP problem was used to generate a FKPP instance.

6.2.1 KP instances

To define our benchmark, we used the KP instances introduced by Pisinger [18]. In particular, several classes of instances have been obtained with different types of correlation between profits and weights. The instances generator, publicly available at `www.diku.dk/~pisinger/codes.html`, takes as input the class number, a positive parameter R that defines the range in which weights are generated, and the total number of items n.
For our experiments we considered only the six classes (11, 12, 13, 14, 15, 16) that are denoted as *hard* in [18] for KP algorithms, and used five different values of R (namely, $R = 10^3, 10^4, 10^5, 10^6$ and 10^7) and five different values of n ($n = 20, 50, 100, 200$, and 500). It turned out that the generator returned integer overflow when generating instances of class 16 with $R \geq 10^5$; thus, we disregarded the corresponding instances. For each class, R and n, we generated one problem, producing a set of 135 KP instances.

6.2.2 Penalty functions

As to the penalty functions, we tested continuous functions expressed by polynomials with degree at most 2, i.e., *linear* or *quadratic* penalty functions. Recall that $\Delta_j^0 = f_j(0)$ and $\Delta_j^1 = f_j(1)$ represent, for a given item j, the value of the penalty function for $x_j = 0$ and $x_j = 1$, respectively. Thus, the general form of the penalty function is

$$f_j(x_j) = -k_j x_j^2 + (\Delta_j^1 - \Delta_j^0 + k_j)x_j + \Delta_j^0 \tag{15}$$

where k_j is a parameter that defines the slope of the function. Given (15), the profit function for each item j reads as follows

$$F_j(x_j) = \begin{cases} 0 & \text{if } x_j = 0 \\ p_j & \text{if } x_j = 1 \\ p_j x_j + k_j x_j^2 + (\Delta_j^0 - \Delta_j^1 - k_j)x_j - \Delta_j^0 & \text{otherwise} \end{cases} \qquad (j \in N) \tag{16}$$

The linear case. When $k_j = 0$, $\forall j \in N$, all profit functions are linear, and all methods described in Sections 3 and 4 can be used to determine an optimal solution. In this case, we consider three different penalty functions, depending on the values Δ_j^0 and Δ_j^1, as follows:

1. *Constant* penalty: if $\Delta_j^0 = \Delta_j^1 = \Delta_j$, the penalty function for item j is constant, and the associated profit function is given by

$$F_j(x_j) = \begin{cases} 0 & \text{if } x_j = 0 \\ p_j & \text{if } x_j = 1 \\ p_j x_j - \Delta_j & \text{if } 0 < x_j < 1 \end{cases} \qquad (j \in N)$$

2. *Increasing* penalty: if $\Delta_j^0 = 0$ and $\Delta_j^1 > 0$, the penalty function for item j is negligible for small fractions x_j while it assumes its maximum for values of x_j close to 1. In this case the profit function is not continuous in $x_j = 1$:

$$F_j(x_j) = \begin{cases} p_j & \text{if } x_j = 1 \\ p_j x_j - \Delta_j^1 x_j & \text{if } 0 \le x_j < 1 \end{cases} \qquad (j \in N)$$

3. *Decreasing* penalty: if $\Delta_j^0 > 0$ and $\Delta_j^1 = 0$, the penalty function for item j assumes its maximum value for $x_j = 0$ and decreases to 0 for $x_j = 1$. In this case the profit function is not continuous in $x_j = 0$:

$$F_j(x_j) = \begin{cases} 0 & \text{if } x_j = 0 \\ p_j x_j - \Delta_j^1 x_j & \text{if } 0 < x_j \le 1 \end{cases} \qquad (j \in N)$$

The quadratic case. If $k_j \neq 0$ the objective function (16) is quadratic. In particular, it corresponds to a convex function if $k_j > 0$, whereas it is a concave function if $k_j < 0$. In the first case Theorem 1 applies, thus both model $MINT$ and the DP algorithm can be used to derive an optimal solution. In this case, however, we could not use model $MGEN$ since our solver does not support the maximization of convex functions. In the concave case, instead, model $MINT$ and the DP algorithm do not provide an optimal solution, whereas model $MGEN$ asks for the minimization of a convex quadratic integer program, which can be tackled by our solver.

Figures 2 and 3 show the different shapes of the profit function in the linear and quadratic cases, respectively, inside the interval $[0, 1]$. They are compared with the linear profit function corresponding to CKP.

Our benchmark includes, for each KP instance, 5 FKPP problems defined according to the penalty functions described above: convex, concave, constant, increasing and decreasing. In all cases, but for the one with increasing penalty, we set $\Delta_j^0 = 0.1p_j$ for each item j. Similarly, all instances but those with decreasing penalty shape, $\Delta_j^1 = 0.1p_j$ for each item j. For the quadratic concave case, we set $k_j = 0.4p_j$, as this choice defined a curve that is a tangent to the linear function $p_j x_j$ for $x_j = 0.5$. By analogy, we set $k_j = -0.4p_j$ in the convex case.

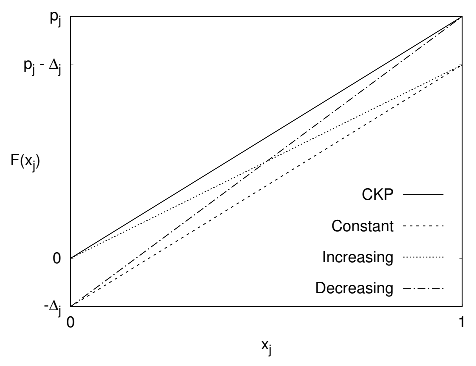

Figure 2: Linear profit functions of FKPP compared with CKP.

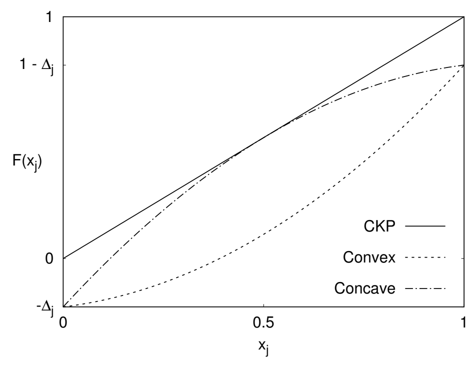

Figure 3: Convex and concave profit functions of FKPP compared with CKP.

6.3 Results on linear instances

Table 1 reports the results for the considered FKPP instances with linear penalty functions ($k_j = 0$). The first and the second columns of the table report the range R and the number of items n of the instances, respectively. Each row summaries the results of eighteen different instances (fifteen for $R \geq 10^5$): one instance for each class (as in [18]) and for each type of linear penalty function (constant, increasing and decreasing). Then the table is vertically divided into six sections, each associated with a model or an algorithm. For each algorithm we report the percentage of instances solved to optimality and the average computing time (for instances solved to optimality only). As algorithms $IDP1$ and $IDP2$ can solve all instances to proven optimality, we report only the associated computing times. Finally, row Avg collects the averages of the above values.

Computational experiments show that the DP algorithms have a much better performance than the direct application of the ILP solver on model $MGEN$: the computing times for the model are often orders of magnitude larger than those of the DP algorithms, and the number of instances solved to optimality is slightly over the 80%. The improved DPs ($IDP1$ and $IDP2$) are, on average, the best performing algorithms for the larger instances having 500 items. For these instances, the computing times are one order of magnitude smaller than those of the corresponding non-improved versions. In addition, $IDP1$ and $IDP2$ are the only methods that can solve all the instances within the time limit. Finally, as to $MCKP$, results show that the performance of this algorithm are only marginally affected by the number of items, whereas it strongly deteriorates when increasing the maximum size R of the items: actually, $MCKP$ is unable to solve any instance with $R = 10^7$.

Figures 4, 5 and 6 report the same results, for constant, increasing and decreasing penalty functions, respectively, using performance profiles (in logarithmic scale). Following the guidelines suggested by Dolan and Moré [9], performance profiles are defined as follows. Let m be any solution method and i denote an instance of the problem. In addition let $t_{i,m}$ be the time required by method m to solve instance i. We define *performance ratio* for pair (i, m) as

$$r_{i,m} = \frac{t_{i,m}}{\min_{m \in M}\{t_{i,m}\}}$$

where M is the set of the considered methods. Then, for each method $m \in M$, we define:

$$\rho_m(\tau) = \frac{|\{i \in I : r_{i,m} \leq \tau\}|}{|I|}$$

		MGEN		*DP1*		*IDP1*	*DP2*		*IDP2*	*MCKP*	
R	*n*	*% Opt.*	*Time*	*% Opt.*	*Time*	*Time*	*% Opt.*	*Time*	*Time*	*% Opt.*	*Time*
10^3	20	100.00	0.01	100.00	0.00	0.00	100.00	0.00	0.00	100.00	0.00
10^3	50	100.00	0.11	100.00	0.01	0.00	100.00	0.00	0.00	100.00	0.00
10^3	100	100.00	1.10	100.00	0.03	0.00	100.00	0.02	0.01	100.00	0.00
10^3	200	100.00	7.30	100.00	0.08	0.01	100.00	0.06	0.01	100.00	0.00
10^3	500	100.00	10.00	100.00	1.06	0.10	100.00	1.31	0.13	100.00	0.02
10^4	20	100.00	0.01	100.00	0.01	0.01	100.00	0.00	0.00	100.00	0.29
10^4	50	100.00	0.31	100.00	0.06	0.02	100.00	0.01	0.01	100.00	0.04
10^4	100	100.00	42.85	100.00	0.22	0.05	100.00	0.07	0.03	100.00	0.05
10^4	200	77.78	1208.05	100.00	0.87	0.13	100.00	0.37	0.06	100.00	0.10
10^4	500	55.56	0.49	100.00	4.66	0.46	100.00	3.48	0.37	94.44	0.20
10^5	20	100.00	0.01	100.00	0.10	0.04	100.00	0.00	0.00	100.00	22.72
10^5	50	100.00	0.06	100.00	0.40	0.13	100.00	0.01	0.01	100.00	29.42
10^5	100	100.00	0.75	100.00	1.58	0.35	100.00	0.33	0.14	100.00	62.46
10^5	200	80.00	0.97	100.00	6.21	0.92	100.00	2.92	0.66	100.00	28.97
10^5	500	60.00	76.46	100.00	38.47	3.45	100.00	27.86	3.09	100.00	0.76
10^6	20	100.00	0.01	100.00	0.95	0.40	100.00	0.00	0.01	100.00	831.63
10^6	50	100.00	0.27	100.00	5.87	1.80	100.00	0.07	0.06	93.33	1102.64
10^6	100	100.00	0.85	100.00	21.61	4.25	100.00	3.86	1.68	86.67	256.69
10^6	200	80.00	1.21	100.00	84.95	12.10	100.00	33.94	8.32	100.00	143.40
10^6	500	86.67	673.10	100.00	531.08	47.37	100.00	395.27	42.46	86.67	466.65
10^7	20	100.00	0.01	100.00	12.10	5.35	100.00	0.04	0.08	0.00	-
10^7	50	100.00	0.37	100.00	78.15	22.85	100.00	0.20	0.19	0.00	-
10^7	100	100.00	1.96	100.00	298.58	60.47	100.00	18.08	12.98	0.00	-
10^7	200	100.00	2.63	100.00	1161.82	167.09	100.00	477.00	129.27	0.00	-
10^7	500	100.00	12.90	0.00	-	650.15	60.00	0.22	701.85	0.00	-
Avg		93.60	81.67	96.00	93.70	39.10	98.40	38.61	36.06	78.44	147.30

Table 1: Average computing time (seconds) over 6 classes of instances with linear profit function.

where I is the set of the instances. Intuitively, $r_{i,m}$ denotes the worsening (with respect to computing time) incurred when solving instance i using method m instead of the best possible one, whereas $\rho_m(\tau)$ gives the percentage of instances for which the computing time of method m was not larger than τ times the time of the best performing method.

The performance profiles clearly show that the DP algorithms are not influenced by the considered penalty function: they “sample” the value of the profit function for integer weight values and associate the profit to the corresponding item fraction. Instead, the performances of the other two methods depend on the penalty function: model $MGEN$ has a good performance for about 40% of the instances with increasing penalty functions, for which it is the fastest method, but then it struggles in solving the instances with constant penalty function. On the contrary, $MCKP$ turns out to be the fastest method for almost 40% of the instances with constant penalty, but it has low performance in the remaining two cases. Among DP algorithms, $IDP2$ turns out to be the most efficient method: actually, this is the fastest algorithm for almost 30% of the instances and turns out to be the best algorithm for all the hard instances that require a large computing time.

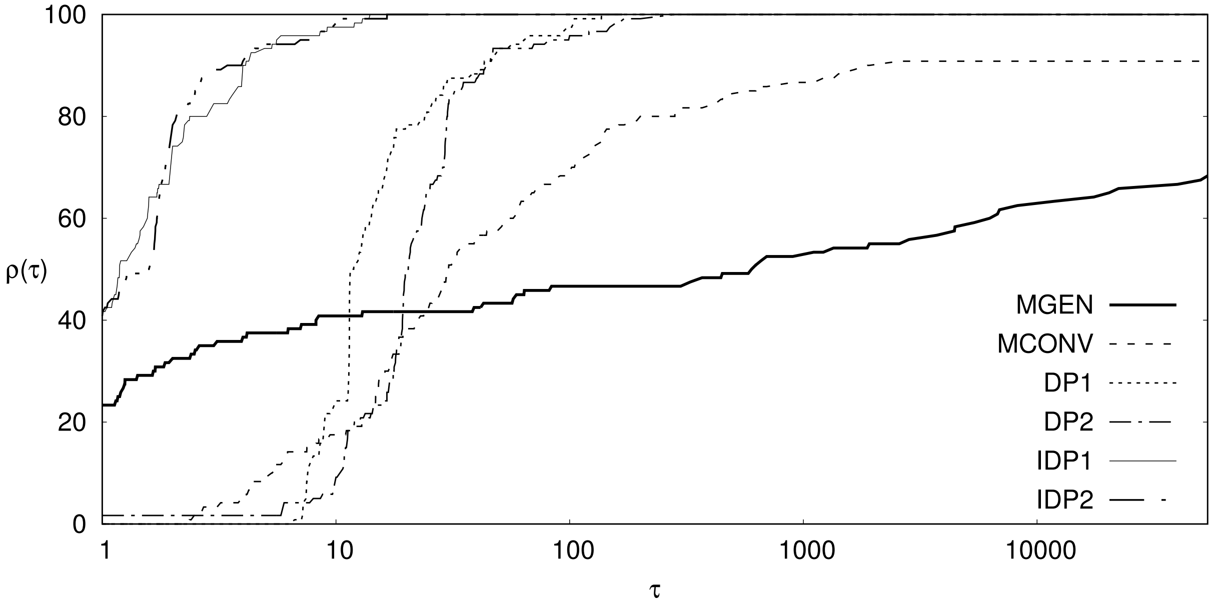

Figure 4: Performance profile of exact methods for FKPP - Constant penalty function.

6.4 Results on convex instances

Table 2 reports the results for the considered FKPP instances with convex profit function ($k_j < 0,\ j \in N$). The table is organized as Table 1 though it does not include results for model $MGEN$, that cannot be optimized using our MIP solver.

The results are somehow similar to those of the linear case, and confirm that the DP algorithms are not really dependent on the shape of the profit function. Conversely, algorithm $MCKP$ has a much more unpredictable behavior that, in any case, deteriorates when increasing the value of R.

Figure 7 reports the performance profile for these instances, showing that $DP2$, $IDP2$ and $MCKP$ are the best methods for 30% of the instances each, while $IDP1$ being the fastest method for 10% of the instances. The fact that $DP2$ can be the best method, i.e., even better that its

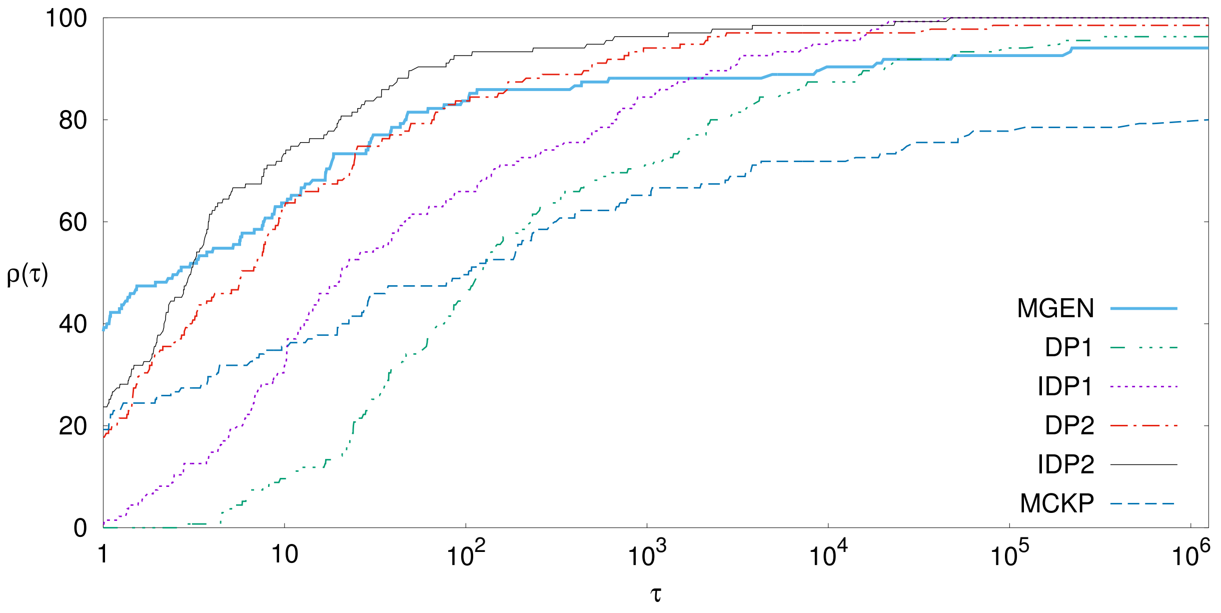

Figure 5: Performance profile of exact methods for FKPP - Increasing penalty function.

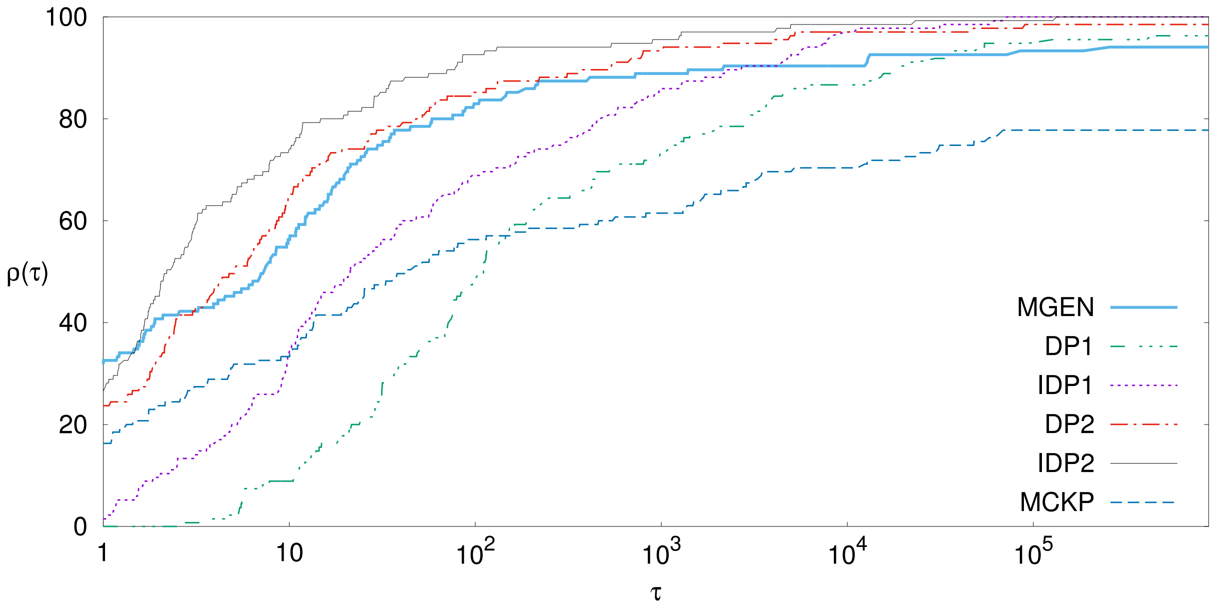

Figure 6: Performance profile of exact methods for FKPP - Decreasing penalty function.

		DP1		*IDP1*	*DP2*		*IDP2*	*MCKP*	
R	*n*	*% Opt.*	*Time*	*Time*	*% Opt.*	*Time*	*Time*	*% Opt.*	*Time*
10^3	20	100.00	0.00	0.00	100.00	0.00	0.00	100.00	0.00
10^3	50	100.00	0.01	0.00	100.00	0.00	0.00	100.00	0.00
10^3	100	100.00	0.03	0.00	100.00	0.02	0.00	100.00	0.00
10^3	200	100.00	0.08	0.01	100.00	0.07	0.02	100.00	0.00
10^3	500	100.00	1.09	0.12	100.00	1.30	0.15	100.00	0.01
10^4	20	100.00	0.01	0.00	100.00	0.00	0.00	100.00	0.16
10^4	50	100.00	0.05	0.02	100.00	0.01	0.01	83.33	0.01
10^4	100	100.00	0.22	0.04	100.00	0.07	0.04	100.00	0.02
10^4	200	100.00	0.79	0.12	100.00	0.36	0.07	100.00	0.04
10^4	500	100.00	4.67	0.48	100.00	3.50	0.36	100.00	0.11
10^5	20	100.00	0.08	0.03	100.00	0.00	0.00	100.00	17.64
10^5	50	100.00	0.39	0.17	100.00	0.01	0.01	80.00	6.14
10^5	100	100.00	1.59	0.36	100.00	0.32	0.15	80.00	0.37
10^5	200	100.00	6.16	0.97	100.00	2.96	0.66	100.00	0.32
10^5	500	100.00	38.44	3.47	100.00	28.52	3.11	60.00	0.35
10^6	20	100.00	0.94	0.40	100.00	0.01	0.01	60.00	1006.89
10^6	50	100.00	5.30	1.60	100.00	0.08	0.05	80.00	421.53
10^6	100	100.00	20.74	4.09	100.00	3.18	1.54	80.00	3.80
10^6	200	100.00	83.88	12.00	100.00	31.16	8.32	100.00	17.54
10^6	500	100.00	593.05	52.29	100.00	507.75	43.01	80.00	54.57
10^7	20	100.00	11.61	5.17	100.00	0.05	0.08	0.00	-
10^7	50	100.00	71.74	21.21	100.00	0.17	0.20	0.00	-
10^7	100	100.00	282.98	56.63	100.00	16.62	13.03	0.00	-
10^7	200	100.00	1114.09	160.22	100.00	357.51	129.78	0.00	-
10^7	500	0.00	-	678.84	60.00	0.23	701.45	0.00	-
Avg		96.00	93.25	39.93	98.40	38.16	36.08	72.13	76.47

Table 2: Average computing time (seconds) over 6 classes of instances with convex objective function. Observe that the times required by the different DP algorithms are almost identical to those of the linear case.

improved counterpart $IDP2$, is due to implementation details that produce some slowdown in the latter; typically these effects are negligible, but they are evident for easy instances that are solved within fractions of a second.

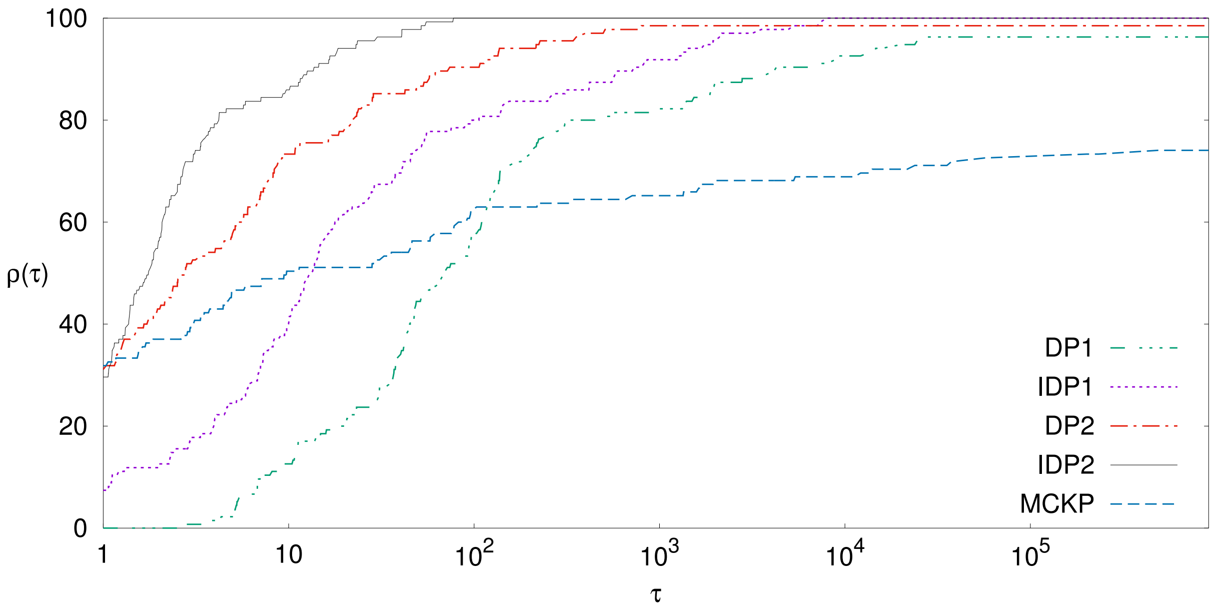

Figure 7: Performance profile of exact methods for FKPP - Quadratic convex penalty function.

6.5 Results on concave instances

As to concave functions, the only available method for computing an optimal solution is the application of our solver to model $MGEN$. Table 3 reports the computing times and the percentage of instances solved to optimality for these instances.

	$n = 20$		$n = 50$		$n = 100$		$n = 200$		$n = 500$	
R	*% Opt.*	*Time*	*% Opt.*	*Time*	*% Opt.*	*Time*	*% Opt.*	*Time*	*% Opt.*	*Time*
10^3	100.00	0.75	83.33	614.84	33.33	0.44	33.33	5.74	0.00	-
10^4	100.00	1.32	66.67	0.94	33.33	0.66	33.33	6.24	16.67	165.84
10^5	100.00	0.93	80.00	19.21	40.00	0.68	20.00	3.99	20.00	29.63
10^6	100.00	0.81	80.00	77.98	20.00	1.06	20.00	4.99	20.00	13.16
10^7	100.00	14.39	40.00	287.85	20.00	4.12	20.00	5.14	20.00	10.93
Avg	100.00	3.64	70.00	200.16	29.33	1.39	25.33	5.22	15.33	54.89

Table 3: Average times (seconds) over 6 classes of instances with quadratic concave objective function.

The results in Table 3 show that solving the problem with concave profit is much more challenging than in the linear case: $MGEN$ is able to consistently solve all the instances with 20 items, with an average computing time of 3.64 seconds, but it fails in solving 30% of the instances with $n = 50$. Remind that, in the linear case, the same algorithm was able to solve all the instances with $n \leq 100$. Results are even worse for larger instances: only 15% of the instances with 500 items are solved to proven optimality within the given time limit.

6.6 Results of heuristic algorithms

Finally, we report the results obtained by executing the heuristic algorithms of Section 5 on our instances. We compare these heuristics with a trivial approach that solves the associated KP instance and takes no item at a fractional level. All these algorithms require a negligible computing time for the considered instances, thus times are not reported in what follows.

We evaluate the quality of a heuristic solution as follows

$$\%prof = 100 \times \frac{z^H}{z^*} \tag{17}$$

where z^* and z^H denote the optimal and the heuristic solution values, respectively.
Figure 8 and 9 plot the quality of the heuristic solutions for all the instances with linear and convex profit functions, as for all these instances the optimal solution value is known. The two figures show the trend with increasing number of items and range, respectively.

From Figure 8, we see that $H1$ finds the best solutions, on average, for instances with 20 items; for larger instances the best algorithm is $H2$. $H3$ has a trend which is similar to that of $H2$, though it performs better than the latter on instances with 20 items only. Finally we observe that the quality of the heuristic solutions improves when increasing the number of items: for $n = 20$, the best performing algorithm (namely, $H1$) has an average profit that is about 1% smaller than the optimal one, while for $n = 500$ this gap is considerably reduced, and the profits of the heuristic solutions are almost the same as the profits of the optimal solutions. This is due to the fact that, for large number of items, the optimal KP solution value is very close to the optimal FKPP solution value (see the plot of algorithm KP), hence the way the spare capacity is filled becomes less crucial. From the practical viewpoint this is good news: for smaller instances, it is important to compute optimal solutions, which can be obtained in short computing time; for large instances, where computing optimal solutions may be time consuming, heuristic solutions are indeed near-optimal. Figure 9 (increasing range) shows a similar behavior concerning the KP solutions. However, in this case, the trend associated with the other heuristic solutions is much more irregular. This plot points out the obvious dominance of $H1$ over KP. In addition, also in this case $H2$ and $H3$ have similar performances, that are better than $H1$ for instances with small R, but become even worse than KP for instances with $R \geq 10^5$.

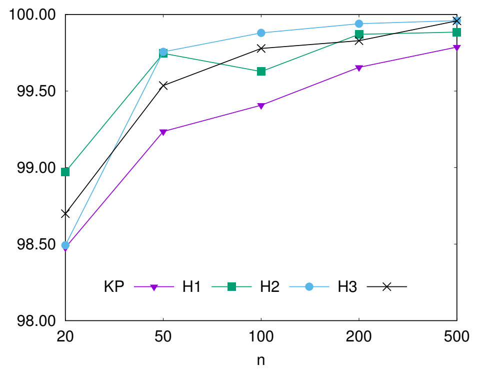

Figure 8: Values of $\%prof$ of the heuristic solutions with increasing number of item (linear and convex profit functions).

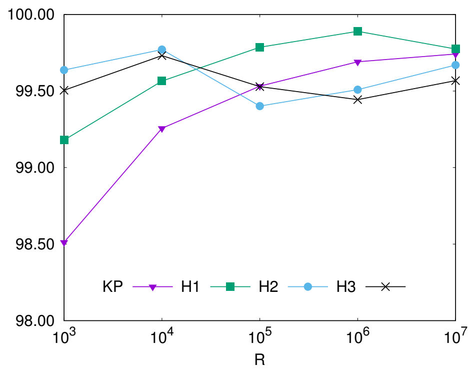

Figure 9: Values of $\%prof$ of the heuristic solutions with increasing range (linear and convex profit functions).

As to instances with concave profit functions, we could solve to optimality only instances with 20 items. Table 4 reports, for the KP and the three heuristic algorithms, the average values of $\%prof$, and the percentage of instances for which each algorithm computed an optimal solutions.

	KP		*H1*		*H2*		*H3*	
R	*% Opt.*	*%prof*	*% Opt.*	*%prof*	*% Opt.*	*%prof*	*% Opt.*	*%prof*
10^3	16.67	95.77	33.33	96.95	16.67	98.88	33.33	99.72
10^4	0.00	96.76	0.00	97.30	33.33	99.53	50.00	99.98
10^5	20.00	99.19	20.00	99.74	20.00	98.54	40.00	99.96
10^6	40.00	98.45	40.00	99.42	20.00	99.37	60.00	99.99
10^7	20.00	99.15	20.00	99.15	20.00	99.23	40.00	99.98
Avg	19.33	97.86	22.67	98.51	22.00	99.11	44.67	99.93

Table 4: Average percentage profit and optimal solutions for the heuristic algorithms, concave profit function (20 items).

In this case the best algorithm is clearly $H3$, which finds an optimal solution in about 44% of the cases and gives, on average, a profit which is 99.93% times the optimal one.

7 Conclusions

We considered integer optimization problems in which the integrality of the variables can be relaxed at the expenses of some penalty in the objective function, a situation that has relevant applications in many contexts. Different from previous approaches from the literature, that always considered the case in which the penalty is constant, we allow the penalty to be dependent on the fractional quantity. As a case study, we concentrated on the Knapsack Problem (KP), which is the simplest integer optimization problem and for which many relevant properties are known. Introducing a penalty associated with fractional variables, we thus defined the Fractional Knapsack Problem with Penalties (FKPP), that we modelled using two mathematical formulations. We studied some properties of the problem, and analyzed the performances of the two models depending on the penalty functions used. From an approximability perspective, we showed that FKPP admits a Fully Polynomial Time Approximation Scheme (FPTAS), independently from the form of the penalty function. From an algorithmic point of view, we introduced two dynamic programming algorithms and three fast heuristic algorithms. We performed an extensive computational analysis to illustrate the performances of all the approaches on a large benchmark set derived from the literature. Our experiments show that we are able to efficiently solve medium-sized FKPP instances, and that the hardness of the instances is closely related to the shape of the penalty function used. Moreover, it seems that FKPP instances are consistently much harder that their KP counterparts, which can be solved in an almost negligible time using combinatorial algorithms from the literature. This suggests that some combinatorial algorithm tailored for FKPP might allow the solution of much larger instances in a reasonable computing time.

Acknowledgments

Enrico Malaguti and Michele Monaci are supported by the Air Force Office of Scientific Research, grants FA9550-17-1-0025 and FA9550-17-1-0067, respectively.
Ulrich Pferschy was supported by the University of Graz project “Choice-Selection-Decision”.

References

[1] C. Archetti and M. G. Speranza, *Vehicle routing problems with split deliveries*, International Transactions in Operational Research, 19 (2012), pp. 3–22.

[2] R. Bellman, *Dynamic Programming*, Princeton University Press, 1957.

[3] K. M. Bretthauer and B. Shetty, *The nonlinear knapsack problem–algorithms and applications*, European Journal of Operational Research, 138 (2002), pp. 459–472.

[4] B. Byholm and I. Porres, *Fast algorithms for fragmentable items bin packing*, Tech. Report 1181, Turku Centre for Computer Science, 2017.

[5] M. Casazza and A. Ceselli, *Mathematical programming algorithms for bin packing problems with item fragmentation*, Computers & Operations Research, 46 (2014), pp. 1–11.

[6] M. Casazza and A. Ceselli, *Exactly solving packing problems with fragmentation*, Computers & Operations Research, 75 (2016), pp. 202–213.

[7] A. Ceselli and G. Righini, *An optimization algorithm for a penalized knapsack problem*, Operations Research Letters, 34 (2006), pp. 394–404.

[8] F. Della Croce, U. Pferschy, and R. Scatamacchia, *Dynamic programming algorithms, efficient solution of the LP-relaxation and approximation schemes for the penalized knapsack problem*, Tech. Report 2017-03-5880, Optimization Online, 2017.

[9] E. D. Dolan and J. J. Moré, *Benchmarking optimization software with performance profiles*, Mathematical Programming, 91 (2002), pp. 201–213.

[10] R. Freling, E. Romeijn, D. R. Morales, and A. Wagelmans, *A branch and price algorithm for the multi-period single-sourcing problem*, Operations Research, 51 (2003), pp. 922–939.

[11] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack Problems*, Springer, Berlin Heidelberg, 2004.

[12] A. Lodi, S. Martello, M. Monaci, C. Cicconetti, L. Lenzini, E. Mingozzi, C. Eklund, and J. Moilanen, *Efficient two-dimensional packing algorithms for mobile WiMAX*, Management Science, 57 (2011), pp. 2130–2144.

[13] E. Malaguti, R. M. Durán, and P. Toth, *Approaches to real world two-dimensional cutting problems*, Omega, 47 (2014), pp. 99–115.

[14] C. Mandal, P. Chakrabarti, and S. Ghose, *Complexity of fragmentable object bin packing and an application*, Computers & Mathematics with Applications, 35 (1998), pp. 91–97.

[15] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*, John Wiley & Sons, Chichester, New York, 1990.

[16] M. Pinedo, *Scheduling. Theory, Algorithms, and Systems*, Springer-Verlag, New York, 2012.

[17] D. Pisinger, *A minimal algorithm for the multiple-choice knapsack problem*, European Journal of Operational Research, 83 (1995), pp. 394–410.

[18] D. Pisinger, *Where are the hard knapsack problems?*, Computers & Operations Research, 32 (2005), pp. 2271–2284.