Improved dynamic programming and approximation results for the Knapsack Problem with Setups

Ulrich Pferschy[*] Rosario Scatamacchia[‡]

Abstract

We consider the 0–1 Knapsack Problem with Setups (KPS). Items are grouped into families and if any items of a family are packed, this induces a setup cost as well as a setup resource consumption. We introduce a new dynamic programming algorithm which performs much better than a previous dynamic program and turns out to be also a valid alternative to an exact approach based on the use of an ILP solver. Then we present a general inapproximability result. Furthermore, we investigate several relevant special cases which still permit fully polynomial time approximation schemes (FPTASs) and others where the problem remains hard to approximate.

Keywords: 0–1 Knapsack Problem with Setups, Approximation Scheme, Dynamic Programming

1 Introduction

The 0–1 Knapsack Problem with Setups (KPS) is a generalization of the standard 0–1 Knapsack Problem (KP) where items belong to disjoint families and can be selected only if the corresponding family is *activated.* The activation of a family incurs setup costs and a consumption of resource. The former worsen the objective function, the latter decreases the available capacity.

KPS has applications of interest for resource allocation problems characterized by classes of elements, for example in make–to–order production

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

[‡]Dipartimento di Automatica e Informatica, Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Torino, Italy, `d021123@polito.it`

contexts or concerning the management of different product categories (see, e.g., [5]). KPS was introduced in [4] where the case with either positive or negative setup costs and profits of items is considered. The authors propose a pseudo-polynomial time dynamic programming approach and a two–phase enumerative scheme. Considering the pseudo–polynomial time algorithm of [4] and the fact that KP is a special case of KPS, namely in the case where there is only one family, we can state that KPS is weakly NP–hard. In [12], a branch and bound algorithm is proposed for KPS. The algorithm turns out to solve instances with up to 10.000 variables. Nonetheless the approach does not solve several large instances due to memory overflow. In [5] an improved dynamic programming algorithm is proposed. The algorithm outperforms the solver CPLEX 12.5 applied to the integer linear programming model of KPS and handles instances with up to 10.000 items. These instances turn out to be more challenging than the ones proposed in [12].

The current state of the art approach for KPS is an exact method proposed in [7]. The approach relies on an effective exploration of the solution space by exploiting the presence of two levels of variables. It manages to solve to optimality all instances with up to 100.000 variables with limited computational effort.

Also a number of problems closely related to KPS were treated in the literature. In [3], a metaheuristic–based algorithm (cross entropy) is proposed in order to deal with KPS with more than one copy per item (cf. the Bounded Knapsack Problem). In [1], a variant of KPS with fractional items is considered and the authors present both heuristic methods and an exact algorithm based on cross decomposition techniques. In [2], the special case of KPS with no setup capacity consumptions but only setup costs is considered. For this so-called fixed–charge knapsack problem the author proposes both heuristic procedures and an exact branch and bound algorithm. A valueable overview of the literature of various KPS variants is provided in [10], which also devises a branch and bound scheme.

The contribution of this paper is twofold: At first we introduce a new improved dynamic programming algorithm motivated by the connection of KPS to a knapsack problem with precedence constraints. This pseudo–polynomial algorithm can be stated with less involved notation and turns out to outperform the recent dynamic programming approach by [5]. Moreover, it performs comparably to the exact approach proposed in [7] but avoids the use of an ILP-solver.

Secondly, we provide further insights into the problem and derive a number of approximation results. More precisely, we show that no polynomial approximation algorithm can exist for the problem in the general case unless

$\mathcal{P} = \mathcal{NP}$ and we investigate several conditions for deriving fully polynomial time approximation schemes (FPTASs). Thus, we make progress in characterizing the borderline between non-approximability and existence of approximation schemes.

The paper is organized as follows: In Section 2, the linear programming formulation of the problem is introduced. The new dynamic programming algorithm is introduced and computational results are presented in Section 3. Approximation results are discussed in Section 4. Section 5 draws some conclusions.

2 Notation and problem formulation

In KPS a set of N families of items is given together with a knapsack with capacity b. Each family $i \in \{1...N\}$ has n_i items, a non–negative setup cost represented by an integer f_i and a non–negative setup capacity consumption denoted by an integer d_i. The total number of items is denoted by $n := \sum_{i=1}^{N} n_i$. Each item $j \in \{1..n_i\}$ of a family i has a non–negative integer profit p_{ij} and a non–negative integer capacity consumption w_{ij}. The problem calls for maximizing the total profit of the selected items minus the fixed costs of the selected families without exceeding the knapsack capacity b. W.l.o.g. we can assume that $w_{ij} + d_i \leq b$ for all items. Any item violating this condition can be removed from the problem since it can never be part of a feasible solution.

To derive an ILP-formulation we associate with each item j of family i a binary variable x_{ij} such that $x_{ij} = 1$ iff item j of family i is placed in the knapsack. To each family i we associate a binary variable y_i such that $y_i = 1$ iff family i is activated. Now KPS can be formulated as follows:

$$
\begin{aligned}
\text{maximize} \quad & \sum_{i=1}^{N}\sum_{j=1}^{n_i} p_{ij}x_{ij} - \sum_{i=1}^{N} f_i y_i && (1)\\
\text{subject to} \quad & \sum_{i=1}^{N}\sum_{j=1}^{n_i} w_{ij}x_{ij} + \sum_{i=1}^{N} d_i y_i \leq b && (2)\\
& x_{ij} \leq y_i \qquad j = 1,\dots,n_i, \quad i = 1,\dots,N && (3)\\
& x_{ij} \in \{0,1\} \qquad j = 1,\dots,n_i, \quad i = 1,\dots,N && (4)\\
& y_i \in \{0,1\} \qquad i = 1,\dots,N && (5)
\end{aligned}
$$

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

3 A new dynamic programming algorithm for KPS

The algorithm we propose is motivated by a dynamic programming scheme for a special case of the *precedence constraint knapsack problem* (PCKP) which is a variant of the standard knapsack problem. In this section we briefly recall PCKP and its relation to KPS. Then our dynamic programming algorithm is presented and comparative computational results are discussed. We remark that the idea of our dynamic program shares structural elements with the dynamic programming approach devised in [8]. However, our main goal is to offer a very simple and viable alternative to the algorithms for KPS available in the literature. Moreover, we exploit our dynamic programming for deriving the approximation results in Section 4. For this purpose we perform dynamic programming by profits although we could just as well run the dynamic program over all weight values.

3.1 The Precedence Constraint Knapsack Problem (PCKP)

The *precedence constraint knapsack problem* (PKCP) or *partially ordered knapsack problem* (see e.g. [9, ch. 13.2] for an overview) is a generalization of KP which imposes a partial order on the items. This means that for each item there is a (possibly empty) set of predecessors which have to be packed into the knapsack if the item is packed. Formally, we are given a directed acyclic graph $G = (V, A)$ where the vertices of V correspond to the items and the existence of an arc $(i, j) \in A$ means that item j can only be part of a solution if also item i is in the solution.

Note that it makes sense to permit also negative profit values in PKCP since due to the precedence constraints also items with negative profits may be part of an optimal solution. PKCP is NP–hard in the strong sense but it is solvable in pseudo–polynomial time on graphs like *out–trees* and *in–trees*. An out–tree is a directed tree with a distinguished vertex called the *root* such that there is a directed path (which is unique) from the root to every vertex in the tree.

KPS can be easily modeled as a PCKP on an out-tree as follows. First, introduce as a root a dummy item with zero weight and profit which has directed arcs to N vertices each of which representing one family. An item of PCKP corresponding to such a *family vertex* carries the setup capacity consumption of the associated family as its weight and the setup cost as negative profit. The actual knapsack items of KPS are inserted in PCKP as leaves of the tree with an incoming arc emanating from the respective family vertex. Thus, the selection of an item requires the activation of the corresponding family.

A dynamic programming algorithm for PCKP for out-trees, called the *left–right approach*, is due to Johnson and Niemi [8]. The main idea of this profit-based algorithm is a suitable ordering of sub–problems identified by sub–trees which corresponds to the structure of the dynamic programming recursion. The computational complexity of this approach is in $O(n\,UB)$, with UB indicating an upper bound on the objective function value. A variant of this algorithm employing dynamic programming by weights was given by Cho and Shaw [6]. Its running time is in $O(n\,b)$.

3.2 Dynamic Programming for KPS

Let us denote by KPS(i,j) the sub–problem induced by the first $i-1$ families, all their items and the first j items of family i. Also, indicate by $r(i,\bar{p})$ the smallest weight for which a solution of KPS(i,n_i) with a total profit at least $\bar{p}$ exists, namely:

$$r(i,\bar{p}) := \min_S \{w(S) \mid S \text{ is a solution of KPS}(i,n_i) \text{ and } p(S) \geq \bar{p}\,\}$$

Here $w(S)$ and $p(S)$ represent the sum of the weights and profits of the items and families included in a solution set S. The minimum over the empty set will be written as ∞. As a restricted variant of $r(i,\bar{p})$ where a certain family i is forced to be activated we define for a triple $(i,j,\bar{p})$:

$$\begin{aligned} z(i,j,\bar{p}) := \quad & \min_{S'}\{w(S') \mid S' \text{ is a solution of KPS}(i,j) \text{ with family i} \\ & \text{activated and } p(S') \geq \bar{p}\,\} \end{aligned}$$

The idea is to analyze the families one by one according to a proper ordering of the sub–problems. More precisely, we first consider the sub–problem with the first family included without any of its items by calculating $z(1,0,\bar{p})$. After that we consider the possible selection of the items of the family one after the other. The analysis of the contribution of the first family is completed by computing $r(1,\bar{p})$. Then we proceed with the other families until $r(N,\bar{p})$ is determined. In each iteration the smallest weight to reach a certain profit level $\bar{p}$ with the first i families is stored in $r(i,\bar{p})$ and then this information is used to analyze the contribution of the next family.

The related approach for PCKP assumes positive profit values. For KPS we have to consider also the cases with negative profits to cover the activation costs of a family. If a family ends up with a negative total profit, its items would not be included at all. Thus, the minimal profit value p_0 the algorithm has to consider is bounded by the negative maximum setup costs.

$$p_0 = -\max\{f_i \mid i = 1, \ldots, N\} \tag{6}$$

As an upper bound UB for the objective function value we could round down the optimum value of the continuous relaxation of KPS. We remark that the gap between the linear relaxation and the optimal solution value can be arbitrarily large as we will see in Section 4.1, nevertheless the bound should be reasonably close to the optimal value in practice. A possible alternative is to consider the trivial upper bound given by neglecting the capacity constraint and selecting all items of all families: $\sum_{i=1}^{N} \sum_{j=1}^{n_i} p_{ij} - \sum_{i=1}^{N} f_i$.

In an initialization step we compute $r(0, \bar{p})$, namely the case where no family is activated and no positive profit can be obtained:

1. for each $(p_0 \le \bar{p} \le UB)$

$$r(0, \bar{p}) = \begin{cases} 0 & \text{if } \bar{p} \le 0, \\ \infty & \text{otherwise.} \end{cases}$$

After this initialization step, the following recursions are iteratively applied for each family $(1 \le i \le N,\ 1 \le j \le n_i)$:

2.

$$z(i, 0, \bar{p}) = \begin{cases} r(i-1, \bar{p} + f_i) + d_i & \text{if } r(i-1, \bar{p} + f_i) + d_i \le b \\ & \bar{p} + f_i \le UB, \\ \infty & \text{otherwise.} \end{cases}$$

3.

$$z(i, j, \bar{p}) = \begin{cases} \min\{z(i, j-1, \bar{p}), z(i, j-1, k) + w_{ij}\} & \text{if } z(i, j-1, k) + w_{ij} \le b \\ & (k = \max\{p_0, \bar{p} - p_{ij}\}), \\ z(i, j-1, \bar{p}) & \text{otherwise.} \end{cases}$$

4.

$$r(i, \bar{p}) = \min\{r(i-1, \bar{p}), z(i, n_i, \bar{p})\}$$

As far as Rule 2 is concerned, by definition family i must be contained in the solution by contributing to the objective only with its setup cost f_i. Thus Rule 2 evaluates if a profit $\bar{p}$ can be obtained by checking whether the sum of the weight to get a profit $\bar{p}+f_i$ with the first $i-1$ families $(r(i-1,\bar{p}+f_i))$ and the setup weight d_i does exceed b or not. Indeed a value $\bar{p}+f_i$ exceeding UB would lead to an infeasible solution and it is not considered. Likewise Rule 3 evaluates the insertion into the knapsack of the item represented by x_{ij} to get a profit $\bar{p}$. Clearly we must have $\bar{p}-p_{ij} \geq p_0$ in that case. The classical knapsack recursion introduced in Rule 4 simply establishes whether the family i is put into the knapsack or not. The optimal solution for KPS is determined by taking the maximum value $\bar{p}$ for which $r(N,\bar{p}) < \infty$.

To bound the pseudo–polynomial running time of this algorithm for KPS it suffices to observe that the array $z(\cdot)$ is evaluated once for each item and family in combination with every profit value while $r(\cdot)$ is evaluated only for every family and every profit value. This yields a trivial $O(n(UB+|p_0|))$ time bound.

In practice, we can obtain a considerable speed-up by systematically considering upper bounds lower than UB for the sub–problems at hand. This will lead to drastic improvements, especially in the early stages of the algorithm where a limited number of families is involved.

It is not difficult to modify the above recursions for dynamic programming by weights where each array entry represents a solution of maximum profit for a certain weight bound $\bar{w} = 1,\ldots,b$. Without going into details we just state that the resulting time complexity is $O(n\,b)$.

From a practical perspective it is also interesting to analyze different choices regarding the sequence of families and items. In our computational tests we consider – somehow counterintuitively – first the *less efficient* families and the less efficient items within each family. As usual, the efficiency of an item is the ratio of profit over weight. For each family, the efficiency is defined as the ratio between the sum of profits of the items minus the setup cost and the sum of the weights (including setup capacity consumption). This choice aims to obtain small upper bounds in the first steps of the dynamic program thus yielding savings in the total running time.

Historically, space requirements were seen as a major obstacle for applying dynamic programming algorithms to large problem instances. Although today's hardware specifications diminish this issue, it is still relevant to be considered also from a theoretical point of view. It is easy to see that for evaluating $z(i,0,\bar{p})$ and $r(i,\bar{p})$ only the values of $r(i-1,\cdot)$ are required. Hence, we can overwrite the values of $r(i-2,\cdot)$ when we start the iteration for family i and restrict the space requirement for array r to $O(UB+|p_0|)$.

The same argument applies for z: Entries $z(i, j, \bar{p})$ require only $z(i, j-1, \cdot)$ for $j = 1, \ldots, n_i$. Thus the space bound of $O(UB + |p_0|)$ suffices also for z.

Another – often overlooked – aspect concerns the storage of the actual solution sets of packed items. Recall that each dynamic programming entry should contain not only a weight value but also the associated item set (cf. [9, ch. 2.3]). This would increase the space requirements by factor of n (or $\log n$, if a bit representation of item sets is employed). However, we can adapt the general recursive storage reduction principle from [11] (see also [9, ch. 3.3]) to keep the space complexity of $O(n + UB + |p_0|)$ without increasing the running time of $O(n(UB + |p_0|))$ but still reporting the optimal solution set. Note that this storage reduction scheme requires an equipartition of the item set and the independent solution of the two resulting subproblems in every iteration. This is in general impossible for the items of KPS which are structured into families. However, for the correctness of the recursive argument of [11] it suffices to construct in each iteration a partitioning of the items into two sets, each of them containing at least a constant fraction of the current item set. Without going into details, we mention that we can realize a partitioning into two subsets each of them containing at least a quarter of the current item set by either partitioning complete families or partitioning only the items of the largest family if it contains at least one half of the current item set.

3.3 Computational Results

All tests have been conducted on an Intel i5 CPU @ 3.2 GHz with 16 GB of RAM. The code has been implemented in the C++ programming language under the Linux operating system. We considered the instances available in [5]. These instances involve a high level of correlation between profits and weights and are perceived as computationally difficult. Weights w_{ij} are integer uniformly distributed in the range [10,100] and profits $p_{ij} = w_{ij} + 10$. The number of families varies from 5 to 30 and the total number of items n from 500 to 10.000. Within each category, 10 instances were tested. The setup costs and weights of the families are equal to

$$f_i = e_1 \left(\sum_{j=1}^{n_i} p_{ij} \right) \tag{7}$$

$$d_i = e_1 \left(\sum_{j=1}^{n_i} w_{ij} \right), \tag{8}$$

with e_1 uniformly distributed in the interval [0.15, 0.25]. The capacity is $b = 0.5\left(\sum_{i=1}^{N}\sum_{j=1}^{n_i} w_{ij}\right)$ and the cardinality of each family ranges in $[k-k/10, k+k/10]$, with $k = n/N$. We compare our new approach with the dynamic program described in [5] and with the exact approach recently laid out in [7]. The latter relies on the use of an ILP-solver and turns out to be very effective. The results are reported in Table 1 in terms of average and maximum CPU time taken to reach the optimal solutions.

		Exact approach from [7]			Dynamic Progr. from [5]			New Dynamic Programming		
N	n	Average time (s)	Max time (s)	#Opt	Average time (s)	Max time (s)	#Opt	Average time (s)	Max time (s)	#Opt
5	500	0.43	0.72	10	0.31	0.49	10	0.02	0.02	10
	1000	0.51	0.66	10	0.92	1.06	10	0.07	0.08	10
	2500	0.98	1.28	10	5.34	5.71	10	0.54	0.56	10
	5000	1.57	1.84	10	20.81	21.52	10	2.00	2.16	10
	10000	3.03	3.67	10	83.93	85.19	10	8.67	8.98	10
10	500	0.46	0.64	10	1.50	11.32	10	0.01	0.02	10
	1000	0.47	0.67	10	1.27	1.38	10	0.07	0.08	10
	2500	0.84	1.01	10	7.33	7.72	10	0.49	0.50	10
	5000	1.47	1.62	10	29.18	30.52	10	1.79	1.84	10
	10000	3.10	3.48	10	149.73	154.61	10	7.15	7.33	10
20	500	0.61	1.14	10	0.56	0.78	10	0.02	0.02	10
	1000	0.51	0.87	10	2.15	2.63	10	0.07	0.07	10
	2500	0.88	1.40	10	13.01	13.68	10	0.42	0.45	10
	5000	1.58	1.95	10	53.45	54.99	10	1.68	1.70	10
	10000	2.96	3.74	10	346.58	353.68	10	6.69	6.71	10
30	500	1.62	4.73	10	0.76	0.89	10	0.02	0.03	10
	1000	0.87	1.95	10	3.32	3.63	10	0.06	0.07	10
	2500	0.99	1.25	10	19.58	20.20	10	0.45	0.46	10
	5000	1.62	2.59	10	79.76	83.42	10	1.66	1.68	10
	10000	4.82	8.07	10	526.61	549.03	10	6.58	6.74	10

Table 1: KPS benchmark instances (from [5]): time (s) and number of optima.

Our dynamic programming algorithm is capable of finding all optima with limited computational effort: it solves all instances in at most 9 seconds (required for one of the largest instances involving 10.000 items), but usually much less. The approach outperforms the dynamic programming algorithm in [5] and the differences in computational times increase as the size of the instances increases. We remark that tests in [5] were performed on a slightly less performing machine, namely an Intel core TMi3 CPU @ 2.1 GHZ with 2GB of RAM, which is expected to be approximately 1.5 times slower than our machine*. Anyhow, the significant differences, in particular on large instances, illustrate that the speed-up we obtain is much larger than the improvement attributed to the different hardware and underlines the impressive effectiveness of our approach. Notice also that the order of magnitude of costs/profits and setups/weights of the families/items is similar for the instances considered. Therefore the mere fact that our algorithm runs over profits rather than weights is not expected to be significant in explaining the differences of the performances. Moreover our dynamic programming compares favorably to the recent exact approach in [7] (running on a similar machine, that is an Intel i5 CPU @ 3.3 GHz with 4 GB of RAM). Computational performances are fully comparable and our dynamic programming can be regarded as a valuable alternative to this current state of the art method but without the necessity to use an ILP-solver.

4 Approximation results

In this section we investigate the properties of KPS and conditions which allow the construction of fully polynomial time approximation schemes (FPTAS) for the problem. For the standard knapsack problem the performance analysis of simple approximation algorithms, such as the greedy algorithm, often relies on a comparison with the linear relaxation of KP. The following consideration shows that this is not a meaningful strategy for KPS.

4.1 Linear relaxation of KPS

In the linear relaxation of KPS both integrality constraints (4) and (5) are replaced by the inclusion in $[0, 1]$. It was shown in [12] that this linear relaxation has a special structure and can be computed in linear time by a reduction to the linear relaxation of a standard KP as follows.

Consider the items sorted in nonincreasing order of their efficiency p_{ij}/w_{ij}. Then for each family compute the *maximum reachable efficiency*, which is given by the first k_i^* items of family i with:

$$k_i^* = \arg\max_k \frac{\sum\limits_{j=1}^{k} p_{ij} - f_i}{\sum\limits_{j=1}^{k} w_{ij} + d_i} \quad (k = 1, \dots, n_i) \quad \forall\, i = 1, \dots, N \tag{9}$$

*We compared the performances of the CPUs by consulting the following website: http://cpuboss.com/

These items are grouped into a new virtual item for each family with profit and weight equal to $\sum_{j=1}^{k_i^*} p_{ij} - f_i$ and $\sum_{j=1}^{k_i^*} w_{ij} + d_i$ respectively. Then consider an instance of KP consisting of N virtual items and all the original items with capacity b. Its linear relaxation can be solved by applying Dantzig's rule and the optimal solution of this problem yields an optimal solution of the linear relaxation of KPS as well (see [12] for a complete proof).

In an optimal solution at most one variable is fractional, either an original item or a virtual item. In the former case an item $x_{\ell j}$ is fractional while the corresponding family ℓ is selected together with all its items of efficiency greater than $p_{\ell j}/w_{\ell j}$. In the latter case a fractional part of all the most effective items, namely the items x_{hj} with $j \leq k_h^*$, of a specific family h is considered. This reformulation of the problem allows us to show the following result:

Proposition 1. *The difference between the linear relaxation and the optimal solution of KPS can be arbitrarily large.*

Proof. Consider the following KPS instance with $N = 1$ and capacity $b = M + 1$ with M being an arbitrary large integer number: There is $f_1 = M - 1$, $d_1 = M$ and $n_1 = 2$ identical items with $p_{1j} = M$ and $w_{1j} = 1$.

The optimal solution of this problem can pack only one item with total profit $p_{1j} - f_1 = 1$. The linear relaxation distributes the setup weight equally over the two items and sets $y_1 = x_{1j} = \frac{M+1}{M+2}$ which gives a total weight of

$$2\,\frac{M+1}{M+2} + M\frac{M+1}{M+2} = M + 1 = b.$$

This yields a total profit of

$$2M\,\frac{M+1}{M+2} - (M-1)\frac{M+1}{M+2} = (M+1)\frac{M+1}{M+2} \approx M,$$

which can become arbitrarily large compared to the integer optimal solution value of 1. □

Thus, for KPS we can not derive an algorithm with bounded performance guarantee by exploiting the properties of the linear relaxation, as it was done for the standard KP (see e.g. [9]).

4.2 Negative approximation result

We prove here the more general result that no approximation algorithm with constant approximation ratio exists for KPS unless $\mathcal{P} = \mathcal{NP}$.

Theorem 1. *KPS does not have a polynomial time approximation algorithm with fixed approximation ratio unless $\mathcal{P} = \mathcal{NP}$.*

Proof. The theorem is proved by reduction from the Subset Sum Problem (SSP), a well–known NP–complete problem where n items j with weights w'_j (with $j = 1, \dots, n$) and a value W' (with $\sum_{j=1}^{n} w'_j > W'$) are given. The decision version of SSP asks whether there exists a subset of items represented by x^* such that $\sum_{j=1}^{n} w'_j x^*_j = W'$.

We build an instance of KPS with just one family with setup cost and weight $f_1 = d_1 = W' - 1$, profits and weights of the $n_1 = n$ items $p_{1j} = w_{1j} = w'_j$ (with $j = 1, \dots, n$) and capacity $b = 2W' - 1$. The capacity bound implies that for every feasible solution x_{1j} there is $\sum_{j=1}^{n_1} p_{1j} x_{1j} = \sum_{j=1}^{n_1} w_{1j} x_{1j} \le b - d_1 = W'$. Hence, the optimal solution of this KPS instance is bounded by $\sum_{j=1}^{n_1} p_{1j} x_{1j} - (W' - 1) \le 1$. Not activating the family at all yields the trivial solution with profit 0. By integrality of the input data this limits the optimal solution value to 0 or 1, where the latter value can be attained if and only if the Subset Sum Problem has a solution.

Thus if there was a polynomial time approximation algorithm with a bounded approximation ratio, we could decide the Subset Sum Problem just by checking if the approximate solution of KPS is strictly positive. Clearly this is not possible under the assumption that $\mathcal{P} \ne \mathcal{NP}$. $\square$

Note that in the KPS-instance of the above proof all items have identical profits and weights and this applies also to the setup values. Thus, the result of Theorem 1 holds also for the "Subset Sum" variant of KPS.

Given this general inapproximability result, it is interesting to investigate to what extent approximation algorithms can be derived when instances with restricting assumptions are considered. In fact, we try to characterize the border between non-approximability and existence of approximation schemes by analyzing four relevant special cases of KPS.

4.3 Case 1, each family can be packed: $d_i + \sum_{j=1}^{n_i} w_{ij} \le b \quad \forall\, i = 1, \dots, N$

This case considers the situation where each single family can be packed into the knapsack with all its items. We first show that a constant performance guarantee exists under this assumption. Consider the linear relaxation of KPS throughout the reduction to the classical knapsack problem as in Section 4.1. As in KP, we can consider a feasible solution of KPS consisting of those variables set to 1 in the optimal solution of the linear relaxation. Denote by z^{LG} the value of such a solution. Then we introduce the following Greedy algorithm with objective function value z^G.

$$z^G = \max\{z^{LG}, p_{max} = \max_i \left(\sum_{j=1}^{n_i} p_{ij} - f_i \right) \} \tag{10}$$

That is, the Greedy algorithm simply takes the best solution among the one given by the variables set to 1 in the optimal solution of the linear relaxation and the ones provided by packing one family only. In addition, it is straightforward to see that for the optimal solution z^* of KPS the following inequality holds, regardless if the fractional variable of the linear relaxation is an existing item or a virtual item:

$$z^* < z^{LG} + p_{max} \tag{11}$$

Consequently we can easily show that Greedy is a 1/2–approximation algorithm for KPS.

Proposition 2. *Algorithm Greedy has a relative performance guarantee of* 1/2 *and this bound is tight.*

Proof. Consider the approximation ratio $\rho = \frac{\max\{z^{LG}, p_{max}\}}{z^*}$ for an arbitrary instance of KPS. We get with (11):

$$\rho > \frac{\max\{z^{LG}, p_{max}\}}{z^{LG} + p_{max}} \geq \frac{\max\{z^{LG}, p_{max}\}}{2\max\{z^{LG}, p_{max}\}} = \frac{1}{2}$$

We can show that the ratio $\frac{1}{2}$ is tight by considering the following example with $N = 2$ families each with $n_i = 2$ and the following entries:

$$b = 2(M+1);$$
$$f_1 = 1,\ d_1 = 1,\ p_{11} = 1,\ p_{12} = M,\ w_{11} = 1,\ w_{12} = M;$$
$$f_2 = 2,\ d_2 = M,\ p_{21} = M,\ p_{22} = 1,\ w_{21} = 1,\ w_{22} = M;$$

M is an arbitrary large integer value. The *maximum reachable efficiency* of the families is equal to $\frac{M}{M+2}$ for family 1 and $\frac{M-2}{M+1}$ for family 2. Since the first family has a greater efficiency ($\frac{M}{M+2} - \frac{M-2}{M+1} = \frac{M+4}{(M+2)(M+1)} > 0$), the linear relaxation sets $y_1 = x_{11} = x_{12} = 1$ and $y_2 = x_{21} = \frac{M}{M+1}, x_{22} = 0$. Therefore algorithm Greedy considers a solution where only the items of the first family are packed into the knapsack with an approximate solution value $z^G = M$ (since $z^{LG} = p_{max} = M$).

The optimal solution instead involves the selection of the second item of the first family and of the first item of the second family with optimal solution value equal to $2M - 3$, which implies that the approximation ratio can be arbitrarily close to $\frac{1}{2}$ as the value of M increases. $\square$

For the case at hand an FPTAS can also be derived.

Theorem 2. *There is an FPTAS for KPS if each family can be packed into the knapsack.*

Proof. First we apply the well–known scaling technique of the profits adopted for the standard knapsack problem (see e.g. [9]). More precisely, we round down the profits of the items p_{ij} and round up the setup costs of the families f_i by considering a scaling factor K yielding the following scaled values:

$$\tilde{p}_{ij} = \left\lfloor \frac{p_{ij}}{K} \right\rfloor \quad \forall\, i,j \text{ and } \tilde{f}_i = \left\lceil \frac{f_i}{K} \right\rceil \quad \forall\, i \tag{12}$$

Running the dynamic programming depicted in Section 3.2 on the problem with scaled profits and costs we get an optimal solution denoted by the sets of the items and families $\tilde{X}$ and $\tilde{Y}$ respectively. We also indicate by z^A the approximate solution value of KPS with items and families in $\tilde{X}$ and $\tilde{Y}$. Finally we denote an optimal solution of KPS by X^* and Y^* (with value z^*). The following series of inequalities holds:

$$\begin{aligned} z^A &= \sum_{ij\in\tilde{X}} p_{ij} - \sum_{i\in\tilde{Y}} f_i \\ &\geq \sum_{ij\in\tilde{X}} K\left\lfloor \frac{p_{ij}}{K} \right\rfloor - \sum_{i\in\tilde{Y}} K\left\lceil \frac{f_i}{K} \right\rceil \\ &\geq \sum_{ij\in X^*} K\left\lfloor \frac{p_{ij}}{K} \right\rfloor - \sum_{i\in Y^*} K\left\lceil \frac{f_i}{K} \right\rceil \\ &\geq \sum_{ij\in X^*} K\left(\frac{p_{ij}}{K} - 1\right) - \sum_{i\in Y^*} K\left(\frac{f_i}{K} + 1\right) = \\ &= z^* - K\left(|X^*| + |Y^*|\right) \end{aligned} \tag{13}$$

Therefore the dynamic programming has a performance guarantee of $(1-\varepsilon)$ if:

$$K \leq \frac{\varepsilon z^*}{|X^*| + |Y^*|} \tag{14}$$

Considering that the cardinality of both X^* and Y^* is trivially bounded by the total number of items n, we can easily satisfy condition (14) by setting $K = \frac{\varepsilon p_{max}}{2n}$.

The running time is dominated by executing the dynamic program on the scaled items. A simple upper bound on the profit range of the problem with scaled items is given by $n\frac{p_{max}}{K} = 2n^2\frac{1}{\varepsilon}$. This yields an overall running time of $O(n^3\frac{1}{\varepsilon})$, which establishes an FPTAS for KPS. $\square$

4.4 Case 2, bounded setup costs: $f_i < \max(p_{ij}) \quad j = 1, \ldots, n_i \quad \forall\, i = 1, \ldots, N$

We consider now the case in which the setup costs of the families are strictly bounded by the value of their most profitable items. As in the general case, we prove that there is no approximation algorithm unless $\mathcal{P} = \mathcal{NP}$.

Theorem 3. *For KPS with $f_i < \max(p_{ij})$ no polynomial time approximation algorithm with fixed approximation ratio exists unless $\mathcal{P} = \mathcal{NP}$.*

Proof. The theorem is proved again by reduction from the Subset Sum Problem involving n items x_j with weights w'_j (with $j = 1, \ldots, n$) and subset sum W' and employs a refinement of the proof of Theorem 1. Let assume that an approximation ratio $\rho > 0$ exists. We build an instance of KPS with one family with weight $d_1 = 0$, setup cost $f_1 = (\frac{1+\rho}{\rho})(W' - 1)$ and capacity $b = W'$. Consider then $n + 1$ items with weights $w_{1j} = w'_j$, profits $p_{1j} = (\frac{1+\rho}{\rho})w'_j$ for $j = 1, \ldots, n$ and $w_{1(n+1)} = W'$, $p_{1(n+1)} = f_1 + 1$. The optimal solution value of this KPS instance can be:

- $z^* = 1$ if the Subset Sum Problem has answer “No”. In this case the best option is to place into the knapsack only the item $(n+1)$ reaching a profit equal to 1, since the other items can not produce a whole positive profit: $(\frac{1+\rho}{\rho})(\sum_{j=1}^{n} w_{1j}x_{1j} - W' + 1) \leq 0$.
- $z^* = \frac{1+\rho}{\rho} > 1$ if the Subset Sum Problem has answer “Yes”. In such a case including into the knapsack the items corresponding to a feasible solution of the Subset Sum Problem dominates the alternative of packing only the item $(n + 1)$.

Therefore, if the Subset Sum Problem admits a solution, a ρ–approximation algorithm would yield a solution value for KPS $z^A \geq \rho(\frac{1+\rho}{\rho}) > 1$. Similarly to the general case, simply checking that z^A is strictly greater than 1 would decide a NP–complete problem contradicting the assumption that $\mathcal{P} \neq \mathcal{NP}$. $\square$

4.5 Case 3, bounded setup costs: For some constant $k \geq 1$ there is $f_i \leq k \cdot \min(p_{ij})$ $j = 1, \ldots, n_i$ $\forall i = 1, \ldots, N$

This case refers to the situation where the setup costs of each family do not exceed the lowest profit of their items by more than a constant factor. In particular the sub–case where k=1 represents the plausible assumption that every item gives a positive contribution, even on its own. First we compute suitable lower and upper bounds for the optimal solution of KPS. Then we rely on the scaling technique discussed in Case 1 in order to derive an FPTAS.

For each family, we consider all ℓ-tuples of items as possible solutions for $1 \leq \ell \leq k$, which takes $O(n^k)$ time. Moreover, in $O(n^k)$ time we can also compute a $\frac{k+1}{k+2}$–approximation algorithm for the standard knapsack problem induced by selecting family i with capacity equal to $b - d_i$, see [9, Sec. 6.1].

Denote by z_i^{AKP} the approximate solution value of the classical knapsack and by z_i^{AKPS} the corresponding KPS value: $z_i^{AKPS} = z_i^{AKP} - f_i$. Likewise, we indicate the optimal value of the knapsack problem induced by family i by z_i^{OKP} and the corresponding KPS solution value by $z_i^{OKPS} = z_i^{OKP} - f_i$. We can show:

Proposition 3. $z_i^{AKPS} \geq \frac{z_i^{OKPS}}{(k+1)(k+2)} \quad \forall i = 1, \ldots, N$

Proof. We prove our claim by distinguishing if the optimal solution of the standard knapsack consists of at most k or more than k items. In the former case the approximation algorithm for the standard knapsack problem would find the optimal solution and thus $z_i^{AKP} = z_i^{OKP}$ and $z_i^{AKPS} = z_i^{OKPS}$. In the latter case at least $k+1$ items are included in the optimal solution implying that $z_i^{OKP} \geq (k+1)\min_j p_{ij} \geq (k+1)\frac{f_i}{k}$. Then the following inequalities hold:

$$\begin{aligned} z_i^{AKPS} &= z_i^{AKP} - f_i \\ &\geq z_i^{AKP} - \frac{k}{k+1} z_i^{OKP} \\ &\geq \frac{k+1}{k+2} z_i^{OKP} - \frac{k}{k+1} z_i^{OKP} \\ &= \frac{k^2 + 2k + 1 - k^2 - 2k}{(k+1)(k+2)} z_i^{OKP} = \frac{z_i^{OKP}}{(k+1)(k+2)} \\ &\geq \frac{z_i^{OKPS}}{(k+1)(k+2)} \end{aligned}$$

which completes the proof. □

If we take the maximum among the z_i^{AKPS} and denote this value by z^{AKPS}, we have with Proposition 3:

$$z^{AKPS} = \max_i(z_i^{AKPS}) \geq \max_i \left(\frac{z_i^{OKPS}}{(k+1)(k+2)} \right) \geq \frac{z^*}{N(k+1)(k+2)} \qquad (15)$$

Therefore, we obtain upper and lower bounds on the optimal value z^* of KPS:

$$z^{AKPS} \leq z^* \leq N(k+1)(k+2)z^{AKPS} \qquad (16)$$

We remark that other procedures to further narrow the interval around z^* may be devised, but this is out of the scope of the present work. Anyhow given the lower and upper bounds in (16) we can prove that:

Theorem 4. *There is an FPTAS for KPS with $f_i \leq k \cdot \min(p_{ij})$ for any positive constant k.*

Proof. The theorem is proved by simply applying the scaling technique as in Case 1 and by setting $K = \frac{\varepsilon z^{AKPS}}{2n}$, a value that satisfies condition (14). An upper bound on the profit range for the problem with scaled items and families follows from (16), namely $N(k+1)(k+2)z^{AKPS}\frac{1}{K} = 2nN(k+1)(k+2)\frac{1}{\varepsilon}$. Also in this case the dynamic programming algorithm implies an FPTAS with running time complexity $O(n^3\frac{1}{\varepsilon} + n^k)$. □

4.6 Case 4, families of bounded size: $n_i \leq C \quad \forall\, i = 1, \dots, N$

We analyze here the case where the number of items in each family is bounded by a constant value C. Under this assumption, it is convenient to see KPS as a special case of the Multiple Choice Knapsack Problem (MCKP). We recall that – similar to the standard knapsack problem – MCKP calls for maximizing the profits of the items while satisfying the capacity constraint. However, the items belong to disjoint classes and MCKP requires that exactly one item from each class has to be placed into the knapsack. We outline the reduction of KPS to MCKP in the following. Associate to a class of MCKP a family from KPS. Then for each possible subset of items in a given family create an item in the corresponding class of MCKP with:

- a profit equal to the sum of the profits of the subset minus the setup cost of their family;
- a weight equal to the sum of the weights of the subset plus the setup weight of the corresponding family.

An item with profit and weight equal to 0 corresponds to the situation in which the family is not selected in a solution of KPS. Subsets yielding a non–positive profit are discarded. Finally, setting the capacity of MCKP equal to b completes the reduction. Since each class of MCKP has a cardinality bounded by the constant 2^C, the cardinality of the power set of the largest family, the total number of items in MCKP is bounded by $N2^C$. It follows immediately from the corresponding results for MCKP that there is a $1/2-$approximation algorithm for this special case of KPS with running time $O(N2^C)$ and also a $4/5-$approximation in $O(N \log N2^C)$ time. If we consider the FPTAS for MCKP as outlined in [9, ch. 11] with complexity $O(\frac{n'm'}{\varepsilon})$, where n' and m' indicate the number of items and classes respectively, we get an FPTAS for KPS as well with running time complexity $O(\frac{N^2}{\varepsilon}2^C)$. Thus we can state the following theorem:

Theorem 5. *KPS with $n_i \leq C$ for a constant C admits an FPTAS.*

5 Conclusions

In this paper we propose an improved dynamic programming algorithm for the knapsack problem with setups (KPS) by effectively leveraging a method from the literature applied for a related knapsack problem. The algorithm competes very favorably with the best currently known exact solution approaches. We also underline the difficulty of KPS by deriving a general non-approximability result. To gain further insights into the structural difficulty of KPS, we investigate several relevant special cases of KPS arising from certain additional, but plausible restrictions on the input data. We manage to derive several approximation algorithms and resulting fully polynomial approximation schemes (FPTASs). In this way we narrow the gap between approximable (in the sense of existence of an FPTAS) and inapproximable cases. In future work we will investigate possible extensions to other variants of KPS. It would also be interesting to construct new benchmark instances which are even more challenging to solve to optimality and to investigate the theoretical properties of such test instances.

Acknowledgments This work was supported by a fellowship from TIM Joint Open Lab SWARM (Turin, Italy).

References

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

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

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

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

[5] K. Chebil, M. Khemakhem. A dynamic programming algorithm for the knapsack problem with setup, *Computers and Operations Research*, 64, 40–50, (2015).

[6] G. Cho, D.X. Shaw. A depth-first dynamic programming algorithm for the tree knapsack problem, *INFORMS Journal on Computing*, 9, 431–438, (1997).

[7] F. Della Croce, F. Salassa, R. Scatamacchia. An exact approach for the 0–1 knapsack problem with setups, (2015), submitted.

[8] D.S. Johnson, K.A. Niemi. On knapsacks, partitions, and a new dynamic programming technique for trees, *Mathematics of Operations Research*, 8, 1–14, (1983).

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

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

[11] U. Pferschy. Dynamic programming revisited: Improving knapsack algorithms, *Computing*, 63, 419–430, (1999).

[12] Y. Yang, R.L. Bulfin. An exact algorithm for the knapsack problem with setup, *International Journal of Operational Research*, 5, 280–291, (2009).