Computers and Operations Research 90 (2018) 208–220

Contents lists available at ScienceDirect

Computers and Operations Research

journal homepage: www.elsevier.com/locate/cor

Exact approaches for the knapsack problem with setups

Fabio Furini [a], Michele Monaci [b,*], Emiliano Traversi [c]

[a] Université Paris Dauphine, PSL Research University, LAMSADE, 75016 Paris, France
[b] DEI, University of Bologna, 40136 Bologna, Italy
[c] Laboratoire d'Informatique de Paris Nord, Université de Paris 13, 93430 Villetaneuse, France

ARTICLE INFO

Article history:
Received 1 February 2017
Revised 13 September 2017
Accepted 18 September 2017
Available online 20 September 2017

Keywords:
Knapsack problems
Column generation
Relaxations
Branch-and-bound algorithms
Computational experiments

ABSTRACT

We consider a generalization of the knapsack problem in which items are partitioned into classes, each characterized by a fixed cost and capacity. We study three alternative Integer Linear Programming formulations. For each formulation, we design an efficient algorithm to compute the linear programming relaxation (one of which is based on Column Generation techniques). We theoretically compare the strength of the relaxations and derive specific results for a relevant case arising in benchmark instances from the literature. Finally, we embed the algorithms above into a unified implicit enumeration scheme which is run in parallel with an improved Dynamic Programming algorithm to effectively solve the problem to proven optimality. An extensive computational analysis shows that our new exact algorithm is capable of efficiently solving all the instances of the literature and turns out to be the best algorithm for instances with a low number of classes.

1. Introduction

The classical *knapsack problem* (KP) is one of the most famous problems in combinatorial optimization. Given a knapsack capacity C and a set $N = \{1, \dots, n\}$ of items, the j-th having a profit p_j and a weight w_j, KP asks for a maximum profit subset of items whose total weight does not exceed the capacity. KP can be formulated using the following Integer Linear Program (ILP):

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

where each variable x_j takes value 1 if and only if item j is inserted in the knapsack.

KP is NP-hard, although in practice fairly large instances can be solved to optimality within low running time. The reader is referred to Martello and Toth (1990) and Kellerer et al. (2004) for comprehensive surveys on applications and variants of this problem.

In this paper we consider a generalization of KP arising when items are associated with operations that require some setup time to be performed. In particular, there is a given set $I = \{1, \dots, m\}$ of *classes* associated with items, and each item j belongs to a given class $t_j \in I$. A positive *setup cost* f_i is incurred and a positive *setup capacity* s_i is consumed in case items of class i are selected in the solution. Without loss of generality, we assume that all input parameters have integer values. The resulting problem is known in the literature as knapsack problem with setup (KPS).

KPS has been first introduced in the literature by Lin (1998) in a survey of non-standard knapsack problems worthy of investigation. In particular, this variant of KP was listed as it finds many practical application, e.g., when industries that produce several types of products must prepare some machinery related to the production of a certain class of products. In addition, it appears as a subproblem in scheduling capacitated machines, and may be used to model resource allocation problems. Guignard (1993) designed a Lagrangean Decomposition for the setup knapsack problem, that may be seen as a variant of KPS in which the setup cost of each class and the profit associated to each item can take also negative values. The version of the problem in which only the setup cost for each class is taken into account, usually denoted as fixed charge knapsack problem, was addressed by Akinc (2006) and Altay et al. (2008). In particular, the former presents an exact algorithm based on a branch-and-bound scheme, while the latter uses cross decomposition to solve the case in which items can be taken at a fractional level. The problem addressed by Michel et al. (2009) is the multiple-class integer knapsack problem, a special case of KPS in which item weights are assumed to be a multiple of their class weight, and lower and upper bounds on the total weight of the used classes are imposed. For this problem, different ILP formulations were introduced and an effective branch-

* Corresponding author.
E-mail addresses: fabio.furini@dauphine.fr (F. Furini), michele.monaci@unibo.it (M. Monaci), emiliano.traversi@lipn.univ-paris13.fr (E. Traversi).

https://doi.org/10.1016/j.cor.2017.09.019
0305-0548/

and-bound algorithm was designed. A Branch-and-Bound algorithm for KPS was given in Yang and Bulfin (2009). This algorithm was tested on instances with up to 10000 variables, and turned out to be effective mainly for instances where profits and weights are uncorrelated – while it ran out of memory for several large correlated instances. Motivated by an industrial application in a packing industry, KPS was studied by Chebil and Khemakhem (2015); this article presented a basic dynamic programming scheme and an improved version of the algorithm, with a reduced storage requirement, that proved able to solve instances with up to 10000 items and 30 classes. Recently, KPS has also been addressed in Pferschy and Scatamacchia and Della Croce et al. (2017). The former introduces a new dynamic programming algorithm, gives negative results on the approximability of the problem in the general case, and considers some special cases for which fully polynomial time approximation schemes exist. The latter presents an exact approach for KPS based on the solution of several ILP models that turn out to be easy to solve in practice. Computational experiments reported in Della Croce et al. (2017), both on instances from the literature and on a large set of new randomly generated problems, show that, for many classes of problems, this approach is the state-of-the-art for the exact solution of KPS. For what concerns approximate solutions, we mention a recent paper by Khemakhem and Chebil (2016), where a tree search combination heuristic is presented. The algorithm is based on the definition of a truncated tree search, where at each level only potentially good nodes are candidates for further exploration, and is tested on the instances introduced in Chebil and Khemakhem (2015).

Paper contributions The contribution of the paper is twofold, as it embraces both theoretical and computational aspects. We develop linear-time algorithms for the optimal solution of the Linear Programming (LP) relaxation of two Integer Linear Programming formulations of KPS. Computational experiments show that these algorithms produce a considerable speedup with respect to the direct use of a commercial LP solver. In addition, we derive for the first time an effective column generation approach to solve a KPS formulation with a pseudo-polynomial number of variables. Finally, we exploit these fast and strong relaxations within an unified branch-and-bound(-and-price) scheme. By reducing the space complexity of the Dynamic Programming algorithm proposed in Chebil and Khemakhem (2015), we managed to improve its computational performance. Since the new exact algorithms are particularly effective on complementary subsets of KPS instances, in order to obtain the best computational performance, we propose a parallel algorithm which exploits the qualities of all the new exact algorithms. We test our new exact algorithms on a large set of instances proposed in the literature and on a new set of larger randomly generated problems. The outcome of our experiments is that the new approaches are competitive with the state-of-the-art exact algorithms for KPS, though they do not require the use of an ILP solver. In addition, we show that on some classes of instances, a considerable speedup may be obtained with respect to the other algorithms proposed so far in the literature.

In the rest of the paper we will denote by n_i the number of items in each class $i \in I$. We assume that $n_i \geq 2$ for some class $i \in I$ and $m > 1$; otherwise, one could associate the setup capacity and cost to the items, yielding a KP. Without loss of generality, we assume that items are sorted according to their class, i.e., class i includes all items $j \in K_i := [\alpha_i, \beta_i]$, where $\alpha_i = \sum_{k=1}^{i-1} n_k + 1$ and $\beta_i = \alpha_i + n_i - 1$. Moreover, we assume that, within each class, items are sorted according to non-increasing profit over weight ratio, i.e.,

$$\frac{p_j}{w_j} \geq \frac{p_{j+1}}{w_{j+1}} \quad j = \alpha_i, \ldots, \beta_i - 1; \quad i \in I.$$

To avoid pathological situations, we also assume that the cost of each class $i \in I$ is smaller than the total profit of its items, i.e., $f_i < \sum_{j \in K_i} p_j$, since otherwise this class will never be used in any optimal solution. We assume that not all items (and classes) can be selected, i.e., $\sum_{j \in J} w_j + \sum_{i \in I} s_i > C$; otherwise a trivial optimal solution is obtained by taking all items and classes. Finally, we assume that each item $j \in N$ satisfies $w_j + s_{t_j} \leq C$; otherwise item j cannot be inserted in any feasible solution, and can be removed from consideration.

Let us introduce a first numerical example, called *Example 1* and reported in Fig. 1. The optimal solution value of Example 1 is 132 and the corresponding solution takes both items of the second class. This example will be used to demonstrate some properties of the KPS models in the following sections.

The paper is organized as follows: in Section 2 we introduce alternative formulations of KPS and discuss the properties of the associated linear programming relaxations. In Section 3 we give efficient combinatorial algorithms for solving the LP relaxations of the models; these algorithms are embedded into an enumerative algorithm described in Section 4. In Section 5 we discuss some improvements to the dynamic programming algorithm proposed in Chebil and Khemakhem (2015). Section 6 describes a relevant special case of KPS and shows the additional properties of the models in this case. Finally, Section 7 reports an extensive computational experience on the solution of the ILP models (and their relaxations) using our algorithms, and compares their performance with other approaches from the literature, and Section 8 draws some conclusions.

2. Integer linear programming models for KPS

In this section we introduce alternative formulations for KPS and discuss the relation between the associated linear programming relaxations. These formulations and the associated LP relaxations will be computationally tested in Section 7.

2.1. Model M1

A natural model for KPS is obtained by introducing x_j variables that have the same meaning as in (1), and decision variables y associated with item classes: in particular, each variable y_i takes value 1 if and only if some item of class i is included in the solution. The resulting model is as follows

$$\max \sum_{j \in N} p_j x_j - \sum_{i \in I} f_i y_i \tag{2}$$

$$\sum_{j \in N} w_j x_j + \sum_{i \in I} s_i y_i \leq C \tag{3}$$

$$x_j \leq y_{t_j} \quad j \in N \tag{4}$$

$$x_j \in \{0, 1\} \quad j \in N \tag{5}$$

$$y_i \in \{0, 1\} \quad i \in I. \tag{6}$$

The objective function (2) maximizes the total profit of the selected items minus the setup cost of the used classes, whereas constraint (3) takes into account that the sum of the item weights and the class setups must not exceed the capacity. Inequalities (4) force a class to be used whenever some item of the class is selected. Finally, (5)–(6) impose all variables to be binary. It is worth mentioning that constraints (4)-(5) and the objective function force the y variables to be binary; thus, in principle, constraints (6) are