Computers & Operations Research 64 (2015) 40–50

Contents lists available at ScienceDirect

Computers & Operations Research

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

A dynamic programming algorithm for the Knapsack Problem with Setup

Khalil Chebil, Mahdi Khemakhem *

LOGIQ, University of Sfax, Sfax, Tunisia

ARTICLE INFO

Available online 19 May 2015

Keywords:
Knapsack problems
Setup
Dynamic programming
Combination
Production planning

ABSTRACT

The Knapsack Problem with Setup (KPS) is a generalization of the classical Knapsack problem (KP), where items are divided into families. An individual item can be selected only if a setup is incurred for the family to which it belongs. This paper provides a dynamic programming (DP) algorithm for the KPS that produces optimal solutions in pseudo-polynomial time. In order to reduce the storage requirements of the algorithm, we adopt a new technique that consists in converting a KPS solution to an integer index. Computational experiments on randomly generated test problems show the efficiency of the DP algorithm compared to the ILOG's commercial product CPLEX 12.5.

1. Introduction

We will refer to the Knapsack Problem with Setup as KPS. It is described as a knapsack problem with additional fixed setup costs discounted both in the objective function and in the constraints. This problem is particularly prevalent in production planning applications where resources need to be set up before a production run.

Our interest in this model was originally motivated by practical problems at a production project with SOTUVER, a leading manufacturer and a supplier of hollow glass in the Tunisian agro-alimentary glass packing industry. This company produces several types of products, including bottles, flacons, and pots. The most important phase in the manufacturing process is the phase of shaping. In fact, to change the production from one product family to an another, the production machinery must be set up and moulds must be changed in the moulding machine. There is no setup between products in the same family. These changes in the manufacturing process require significant setup time and costs. Accordingly, the company needs to decide on how to choose orders so as to maximize the total profit. This represents a typical case involving a knapsack problem with setup model that can be used to solve this problem.

The Knapsack Problem with Setup is defined by a knapsack capacity $b \in \mathbb{N}$ and a set of N classes of items. Each class $i \in \{1,\ldots,N\}$ is composed of n_i items and characterized by a negative integer f_i and a non-negative integer d_i representing its setup cost and setup capacity consumption, respectively. Each item $j \in \{1,\ldots,n_i\}$ of a class i is labeled by a profit $c_{ij} \in \mathbb{N}^{N\times n_i}$ and a capacity consumption $a_{ij} \in \mathbb{N}^{N\times n_i}$. The aim is to maximize the total profit of the selected items minus the fixed costs incurred for setting-up the selected classes.

The KPS can be formulated by a 0-1 linear program as follows:

$$\text{Max } z = \sum_{i=1}^{N}\sum_{j=1}^{n_i} c_{ij}x_{ij} + \sum_{i=1}^{N} f_i y_i \tag{1}$$

s.t.

$$\sum_{i=1}^{N}\sum_{j=1}^{n_i} a_{ij}x_{ij} + \sum_{i=1}^{N} d_i y_i \le b \tag{2}$$

$$x_{ij} \le y_i \quad \forall i \in \{1,\ldots,N\} \quad \forall j \in \{1,\ldots,n_i\} \tag{3}$$

$$x_{ij}, y_i \in \{0,1\} \quad \forall i \in \{1,\ldots,N\} \quad \forall j \in \{1,\ldots,n_i\} \tag{4}$$

Eq. (1) represents the objective function for KPS. Constraint (2) ensures that the weight of selected items in the knapsack, including their setup capacity consumption, does not exceed knapsack capacity b. Constraints (3) guarantee that each item is selected only if it belongs to a class that has been setup. Constraints (4) require the decision variables to be binary where x_{ij} represents the item variables and y_i the class setup variables. In fact, y_i is equal to 1 if the knapsack is set up to accept items belonging to class i; otherwise it is equal to 0. x_{ij} is equal to 1 if the item j of the class i is placed in the knapsack and is equal to 0 otherwise.

The standard 0-1 Knapsack Problem (KP), which is known to be an NP-hard problem [1], is a variant of KPS wherein the number of item classes is equal to one. The KP has been the subject of extensive research, and a detailed summary of the major works on the issue is available in [1,2].

* Corresponding author.
E-mail addresses: khalil.chebil@issatm.rnu.tn (K. Chebil), mahdi.khemakhem@enetcom.rnu.tn (M. Khemakhem).

http://dx.doi.org/10.1016/j.cor.2015.05.005
0305-0548/

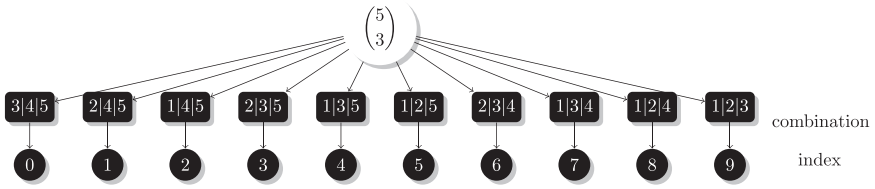

Fig. 1. ⟨3−*combinations*⟩ list of 5.

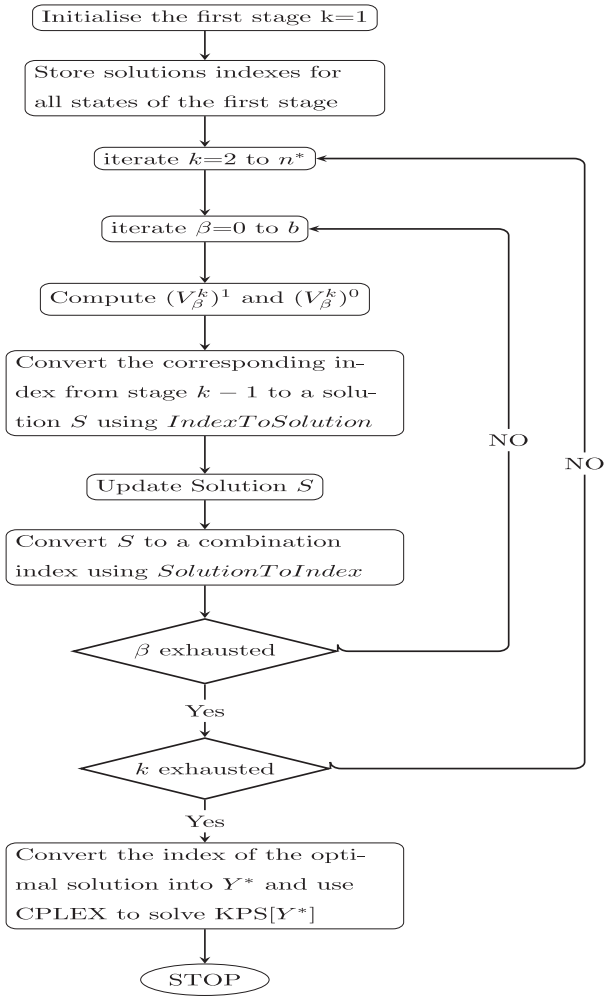

Fig. 2. A sketch of the DPI algorithm.

KPS has similarities to several other knapsack variants in addition to KP. As presented by Chajakis and Guignard [3], the Set-up Knapsack Problem (SKP) occurs as a subproblem of a parallel machine scheduling problem with setups. This problem is similar to KPS, except for the fact that the setup cost of each class and the profit associated to each item can take on real numbers that may be negative or non-negative. A further constraint is added to make sure that if the knapsack is setup for a class, at least one item of this class is selected.

A thorough survey of the literature on two variants of knapsack problems with setups has recently been presented in the work of Michel et al. [4] namely the multiple-class binary knapsack problem with setups (MBKPS) where item weights are assumed to be a multiple of their class weight and the continuous knapsack problems with setups (CKS) where each class holds a single item and a fraction of an item can be selected while incurring a full setup. Michel et al. provided an extension of the branch-and-bound algorithm proposed by Horowitz and Sahni [5] for problems with positive setup costs.

The Fixed Charge Knapsack Problem (FCKP) is a particular case of KPS that includes the setup capacity consumption but not the setup cost, see [6]. The bounded Knapsack Problem with Setups (BKPS), which is a generalization of FCKP wherein a limited copy of each item may be added to the knapsack, was presented in [7,8]. The Integer Knapsack Problem with Set-up Weights (IKPSW), see [9], is a generalization of BKPS in which an unlimited copy of each item may be added to the knapsack. This model has found application in the area of aviation security, see [10,11].

The remainder of this paper is organized as follows: in Section 2, we introduce a basic dynamic programming (*DPB*) algorithm to optimally solve the KPS. We then present an improved version of the *DPB* algorithm (*DPI*) that has the ability to solve large problem instances (up to 10,000 items) in a pseudo-polynomial time. In Section 3, we analyze the computational behavior of these algorithms on a set of randomly generated test problems. Section 4 concludes by a brief summary and directions for future research.

2. A dynamic programming algorithm for the KPS

Dynamic Programming (DP) was introduced by Bellman [12]. The basic idea of DP is to solve a problem by a divide-and-conquer approach wherein the solutions of overlapping subproblems are reused to avoid the recalculation of solutions. Toth [13] and Horowitz and Sahni [5] presented an improved dynamic programming algorithm for solving the knapsack problem. More recently, Pisinger [14] proposed a minimal algorithm for the 0-1 knapsack problem based on dynamic programming. A first hybrid approach combining DP with an enumerative scheme was proposed by Plateau and Elkihel [15]. Generally, dynamic programming-based algorithms are efficient and easy to implement particularly for small and medium-sized instances. Chajakis and Guignard [3] suggested a dynamic programming algorithm and two versions of a two-phase enumerative scheme for the SKP. Experiments were performed only on uncorrelated instances with low range coefficients (i.e. a_{ij} from [1,10]). Since the DP algorithm presented in their paper has a pseudo-polynomial worst case complexity, the large number of coefficients would increase the difficulty of instances and need more storage. In their model, the setup cost of each class and the profit associated with each item can take on real numbers that may be negative or non-negative. The fact that parts of the variables are fixed to 0 by a preprocessing procedure remarkably reduces the size of the problem, thus making instances easier. McLay and Jacobson [9] have also provided a dynamic programming algorithm for IKPSW that produces an optimal solution in a pseudo-polynomial time but is not practical for solving large problem instances.

In this section, we start by providing a brief outline of a DP algorithm for the 0-1 knapsack problem. We then present a number of preliminary considerations in the knapsack problem with setup that are relevant to the design of the proposed algorithm. We also describe the general DP procedure.

2.1. A dynamic programming algorithm for the 0-1 KP

The classical knapsack problem is defined as follows: we consider a set of n items, with each item j having an integer profit