Computers & Industrial Engineering 61 (2011) 274–278

Contents lists available at ScienceDirect

Computers & Industrial Engineering

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

A dynamic programming method with lists for the knapsack sharing problem

V. Boyer *, D. El Baz, M. Elkihel

CNRS, LAAS, 7 avenue du Colonel Roche, F-31077 Toulouse, France
Université de Toulouse, UPS, INSA, INP, ISAE, LAAS, F-31077 Toulouse, France

ARTICLE INFO

Article history:
Available online 3 November 2010

Keywords:
Knapsack sharing problem
Combinatorial optimization
Max–min programming
Dynamic programming

ABSTRACT

In this paper, we propose a method to solve exactly the knapsack sharing problem (KSP) by using dynamic programming. The original problem (*KSP*) is decomposed into a set of knapsack problems. Our method is tested on correlated and uncorrelated instances from the literature. Computational results show that our method is able to find an optimal solution of large instances within reasonable computing time and low memory occupancy.

1. Introduction

The knapsack sharing problem (KSP) is a max–min mathematical programming problem with a knapsack constraint (see Brown, 1979, 1991, 1994). The KSP is NP-complete. The KSP occurs when resources have to be shared or distributed fairly to several entities, e.g. distribution of scarce resources like ammunition or gasoline among different military units (see Brown, 1979) or budget shared between districts of a city (see Yamada, Futakawa, & Kataoka, 1997, 1998). The KSP is composed of n items divided into m different classes. Each class $\mathcal{N}_i$ has a cardinality n_i with $\sum_{i\in\mathcal{M}} n_i = n$ and $\mathcal{M} = \{1, 2, \ldots, m\}$. Each item $j \in \mathcal{N}_i$ is associated with:

- a profit p_{ij},
- a weight w_{ij},
- a decision variable $x_{ij} \in \{0, 1\}$.

We wish to determine a subset of items to be included in the knapsack of capacity C, so that the minimum profit associated with the different class is maximised. The KSP can be formulated as follows:

$$(KSP)\begin{cases} \max\ \min\limits_{i\in\mathcal{M}} \left\{ \sum\limits_{j\in\mathcal{N}_i} p_{ij}\cdot x_{ij} \right\} = z(KSP), \\ s.t.\ \sum\limits_{i\in\mathcal{M}} \sum\limits_{j\in\mathcal{N}_i} w_{ij}\cdot x_{ij} \leqslant C, \\ x_{ij} \in \{0,1\} \text{ for } i \in \mathcal{M} \text{ and } j \in \mathcal{N}_i, \end{cases} \tag{1.1}$$

where $z(KSP)$ denotes the optimal value of the problem (KSP) and for $i \in \mathcal{M}$ and $j \in \mathcal{N}_i$, w_{ij}, p_{ij} and C are positive integers. Furthermore, we assume that $\sum_{i\in\mathcal{M}}\sum_{j\in\mathcal{N}_i} w_{ij} > C$ and $\max_{i\in\mathcal{M}, j\in\mathcal{N}_i}\{w_{ij}\} \leqslant C$.

A common way to solve the KSP consists of its decomposition into knapsack problems (see for examples Hifi & Sadfi, 2002; Yamada et al., 1998). Indeed, for a class $i \in \mathcal{M}$, we define the following problem:

$$(KP_i(C_i))\begin{cases} \max \sum\limits_{j\in\mathcal{N}_i} p_{ij}\cdot x_{ij} = z(KP_i(C_i)), \\ s.t.\ \sum\limits_{j\in\mathcal{N}_i} w_{ij}\cdot x_{ij} \leqslant C_i, \\ x_{ij} \in \{0,1\}\quad j \in \mathcal{N}_i. \end{cases} \tag{1.2}$$

The objective is then to find $(C_1^*, C_2^*, \ldots, C_m^*)$ such that

$$\sum_{i\in\mathcal{M}} C_i^* \leqslant C,$$

and

$$\min_{i\in\mathcal{M}}\{z(KP_i(C_i^*))\} = z(KSP),$$

where for a problem $\mathcal{P}, z(\mathcal{P})$ represents its optimal value. An upper bound and a lower bound of $z(\mathcal{P})$ will be denoted, respectively, by $\bar{z}(\mathcal{P})$ and $\underline{z}(\mathcal{P})$, respectively.

Hifi et al. have proposed to solve the knapsack problems $(KP_i(C_i))_{i\in\mathcal{M}}$ via a dense dynamic programming algorithm in Hifi and Sadfi (2002) and Hifi et al. (2005). Their method starts with $C_i = 0, i \in \mathcal{M}$, and increases regularly the capacities until $\sum_{i\in\mathcal{M}} C_i > C$.

In this article, we propose an algorithm for solving the KSP. Our algorithm is based on a dynamic programming procedure with dominance technique to solve the knapsack problems $(KP_i(C_i))_{i\in\mathcal{M}}$. Our algorithm starts by solving the problems $(KP_i(C_i))_{i\in\mathcal{M}}$ with $C_i \geqslant C_i^*, i \in \mathcal{M}$ and $\sum_{i\in\mathcal{M}} C_i \geqslant C$. At each step, we try to decrease

* Corresponding author. Tel.: +33 6 11 23 24 75.
E-mail addresses: vboyer@laas.fr (V. Boyer), elbaz@laas.fr (D. El Baz), elkihel@laas.fr (M. Elkihel).

0360-8352/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cie.2010.10.015

the values of $(C_i)_{i\in\mathcal{M}}$, towards $(C_i^*)_{i\in\mathcal{M}}$. The use of lists permits one to reduce the memory occupancy; the expected benefit being the solution of large instances.

Without loss of generality, we consider in the sequel that the items in a class $i \in \mathcal{M}$ are sorted according to decreasing ratios $\frac{p_{ij}}{w_{ij}}, j \in \mathcal{N}_i$.

In Section 2, we present the dynamic programming algorithm used to solve the problems $(KP_i)_{i\in\mathcal{M}}$. Section 3 deals with the algorithm we propose for the KSP. Computational results are displayed and analyzed in Section 4. Some conclusions and perspectives are presented in Section 5.

2. Basic dynamic programming procedure

In order to solve the problems $(KP_i(C_i))_{i\in\mathcal{M}}$, we use a dynamic programming algorithm with dominance (see Bellman, 1957; Boyer, El Baz, & Elkihel, 2009, 2010; El Baz & Elkihel, 2005).

2.1. Lists construction

We recall that $i \in \mathcal{M}$ denotes the index of the ith class of (KSP).

A list $\mathcal{L}_{ik}$ is associated with each step $k \in \mathcal{N}_i$:

$$\mathcal{L}_{ik} = \left\{ (w,p) | w = \sum_{j=1}^{k} w_{ij} \cdot x_{ij} \leqslant C_i \text{ and } p = \sum_{j=1}^{k} p_{ij} \cdot x_{ij}, x_{ij} \in \{0,1\},\ j \in \{1,2,\ldots,k\} \right\}. \quad (2.1)$$

The algorithm begins with the lists $\mathcal{L}_{i0} = \{(0,0)\}$.

At each step $k \in \mathcal{N}_i$, the new list $\mathcal{L}_{ik}$ is obtained as follows:

$$\mathcal{L}_{ik} = \mathcal{L}_{i(k-1)} \cup \{(w + w_{ik}, p + p_{ik}) | (w,p) \in \mathcal{L}_{i(k-1)}, w + w_{ik} \leqslant C_i\}.$$

Notation. For simplicity of presentation the above equation will also be written as follows in the sequel.

$$\mathcal{L}_{ik} = \mathcal{L}_{i(k-1)} \oplus \{(w_{ik}, p_{ik})\}.$$

The states (w, p) in a list are sorted according to the decreasing value of p.

From the dynamic programming principle, dominated states, i.e. states (w, p) such that there exists a state (w', p') with $w' \leqslant w$ and $p' \geqslant p$, are removed from the list.

2.2. State elimination via upper bounds

In order to shrink lists $\mathcal{L}_{ik}, k \in \mathcal{N}_i$, an upper bound $\bar{z}(w,p)$, associated with state $(w,p) \in \mathcal{L}_{ik}$, is computed. For this purpose, we solve exactly the following linear continuous knapsack problem via the Martello & Toth's algorithm (see Martello & Toth, 1977):

$$(LP_i^{(w,p)}(C_i)) \begin{cases} \max p + \sum_{j=k+1}^{n_i} p_{ij} \cdot x_{ij}, \\ s.t. \sum_{j=k+1}^{n_i} w_{ij} \cdot x_{ij} \leqslant C_i - w, \\ x_{ij} \in [0,1], j \in \{k+1,\ldots,n_i\}. \end{cases} \quad (2.2)$$

Let $\bar{z}(w,p) = \lfloor z(LP_i^{(w,p)}(C_i)) \rfloor$.

If $\bar{z}(w,p) \leqslant \underline{z}(KSP)$, then the states (w, p) can be discarded.

We shall have:

$$\mathcal{L}_{ik} := \mathcal{L}_{i(k-1)} \oplus \{(w_{ik}, p_{ik})\} - \mathcal{D}_{ik} - \mathcal{B}_{ik},$$

where

- $\mathcal{D}_{ik}$ represents the list of dominated states in $\mathcal{L}_{i(k-1)} \oplus \{(w_{ik}, p_{ik})\}$,
- $\mathcal{B}_{ik}$ represents the list of states in $\mathcal{L}_{i(k-1)} \oplus \{(w_{ik}, p_{ik})\}$ to be eliminated via upper bounds.

In the sequel, this phase is the so-called *NextList* phase.

2.3. Fixing variables

The following two methods are used to fix variables of the problem (KP_i).

2.4. Variable reduction technique 1

Let $i \in \mathcal{M}, k \in N_i$ and $(w,p) \in \mathcal{L}_{ik}$.

If $p > \bar{z}(KSP)$, where $\bar{z}(KSP)$ is an upper bound of (KSP) obtained by the solution of the linear continuous relaxation of (KSP), then all free variables $x_{ij}, j \in \{k+1,\ldots,n_i\}$, can be fixed at 0 for the state (w,p).

Indeed, as $z(KSP) \leqslant \bar{z}(KSP)$, when $p > \bar{z}(KSP)$ we can stop the exploration of this state because it will not give a better optimal value for (KSP).

The second method to fix variables uses information provided by the solution of $(LP_i^{(w,p)}(C_i))$ associated to a state $(w,p) \in \mathcal{L}_{ik}$, $k \in N_i$. We use the following rule to fix the free variables of a state (w, p).

2.5. Variable reduction technique 2 (see Nemhauser & Wolsey, 1988)

Let $i \in \mathcal{M}, k \in N_i$ and $(w,p) \in \mathcal{L}_{ik}$.

Let d be the index of the critical variable of $(LP_i^{(w,p)}(C_i))$, i.e.:

$$\sum_{j=k+1}^{d-1} w_{ij} \leqslant C_i - w \text{ and } \sum_{j=k+1}^{d} w_{ij} > C_i - w.$$

If for $j \in \{k+1,\ldots d-1, d+1,\ldots,n_i\}$, we have

$$\left\lfloor z(LP_i^{(w,p)}(C_i)) - \left| p_{ij} - w_{ij} \cdot \frac{p_{id}}{w_{id}} \right| \right\rfloor \leqslant \underline{z}(KSP),$$

where $\underline{z}(KSP)$ is a lower bound of (KSP), then x_{ij} can be fixed to 1 if $j < d$ and to 0 otherwise. The computation of $\underline{z}(KSP)$ is detailed in the next section.

3. An algorithm for the KSP

In this section, we show how the dynamic programming method presented in the above section can be used to find an optimal solution of (KSP).

For simplicity of notation, $\underline{z}(KSP)$ is denoted by $\underline{z}$ in the sequel.

3.1. The main procedure DPKSP

The principle of our algorithm can be presented briefly as follows:

–A first lower bound of (KSP), $\underline{z}$, is computed with the greedy heuristic the so-called *GreedyKSP* (see Algorithm 1).

–At each step k of the dynamic programming method:

- the lists $(\mathcal{L}_{ik})_{i\in\mathcal{M},\ k\leqslant n_i}$ are computed via the procedure *NextList* presented in the above section,
- then, we try to reduce the free variables associated with each state in $(\mathcal{L}_{ik})_{i\in\mathcal{M},\ k\leqslant n_i}$,
- finally, the lower bound $\underline{z}$ and the capacities $(C_i)_{i\in\mathcal{M}}$, respectively, are updated via the procedures *UpdateZ* and *UpdateC*, respectively described below.