Approximation Schemes for Knapsack Problems with Shelf Divisions *

E. C. Xavier† F. K. Miyazawa†

February 23, 2006

Abstract

Given a knapsack of size K, non-negative values d and Δ, and a set S of items, each item $e \in S$ with size s_e and value v_e, we define a shelf as a subset of items packed inside a bin with total items size at most Δ. Two subsequent shelves must be separated by a shelf divisor of size d. The size of a shelf is the total size of its items plus the size of the shelf divisor. The Shelf-Knapsack Problem (SK) is to find a subset $S' \subseteq S$ partitioned into shelves with total shelves size at most K and maximum value. The Class Constrained Shelf Knapsack (CCSK) is a generalization of the problem SK, where each item in S has a class and each shelf in the solution must have only items of the same class. We present approximation schemes for the SK and the CCSK problems. To our knowledge, these are the first approximation results where shelves of non-null size are used in knapsack problems.

Key Words: Approximation algorithms, approximation schemes, knapsack problem, shelf packing.

1 Introduction

In this paper we present approximation schemes for knapsack problems where items are separated by shelves. We first define these problems formally.

*This research was partially supported by CNPq (Proc. 478818/03-3, 306526/04-2, 471460/04-4, and 490333/04-4), ProNEx–FAPESP/CNPq (Proc. 2003/09925-5) and CAPES.

†Instituto de Computação — Universidade Estadual de Campinas, Caixa Postal 6176 — 13084–971 — Campinas–SP — Brazil, {eduardo.xavier,fkm}@ic.unicamp.br.

Given a knapsack of size K, the size of a shelf divisor d, the maximum size of a shelf Δ, and a set S of items, each item $e \in S$ with size s_e and value v_e, a *shelf* is defined as a subset of items packed inside a bin with total items size at most Δ. The size of a shelf is the total size of its items plus the size of a shelf divisor. We also consider a version where shelves must have size at least δ, where $0 < \delta \leq \Delta$. Since all shelves considered in this paper must have size at most Δ, we use the notation δ-shelf to denote a shelf that must have size at least δ. The Shelf-Knapsack Problem (SK) is to find a subset $S' \subseteq S$ partitioned into shelves with total shelf size at most K and maximum total value. The Class Constrained Shelf Knapsack Problem (CCSK) is a generalization of the SK problem, where each item $e \in S$ has a class c_e and each shelf must have only items of the same class.

The SK and CCSK problems are NP-hard since they are generalizations of the knapsack problem. We note that the term shelf is used under another context in the literature for 2-D packing problems.

There are many practical applications for these problems. For example, when the items to be packed must be separated by non-null shelf divisors (inside a bin) and each shelf cannot support more than a certain capacity. The CCSK problem is adequate when some items cannot be stored in a same shelf (like foods and chemical products). In most of the cases, the sizes of the shelf divisions have non-negligible width. Although these problems are very common in practice, to our knowledge this paper is the first to present approximation results for them.

An interesting application for the CCSK problem, where each shelf must be a δ-shelf, was introduced by Ferreira et al. [5] in the iron and steel industry, where a raw material roll must be cut into final rolls grouped by certain properties after two cutting phases. The rolls obtained after the first phase, called primary rolls, are submitted to different processing operations (tensioning, tempering, laminating, hardening etc.) before the second phase cut. Due to technological limitations, primary rolls have a maximum and minimum allowable width and each cut generates a loss in the roll width.

In Table 1, we present some common characteristics for final rolls. We consider three classes in this example, one for each different thickness. The hardness interval of items with the same thickness are overlapped in a common interval to satisfy all hardness requirements. If there are items for which hardness cannot be assigned to the same thickness class, a new class must be defined for these ones.

In the example of Table 1, we have a raw material roll of size K (1040 mm) that must be cut and processed into the final items. This roll is first cut in three primary rolls according to the items in the three different classes. Each primary roll is processed by different operations to acquire the required thickness and hardness before obtaining the final rolls. Each cutting in the primary roll generates a

Width (mm)	Hardness ($kg \cdot mm^{-2}$)	Thickness (mm)
250	50 to 70	4.50
200	60 to 75	4.50
60	32 to 39	3.50
60	32 to 41	3.50
60	20 to 30	2.50
60	24 to 35	2.50

Table 1: Characteristics of final rolls.

loss due to the width of the cutter knife and the trimming process. This loss corresponds to the size of the shelf divisors. The items are obtained after the second cutting phase. In this application, we only worry about the loss generated in the first cutting phase. The process is illustrated in Figure 1.

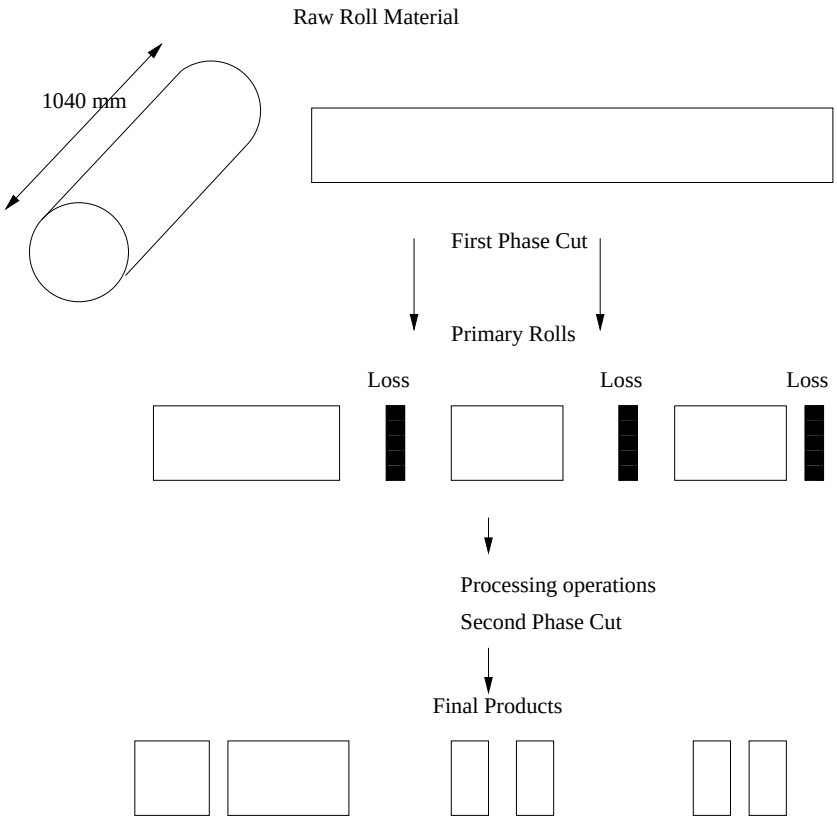

Figure 1: The two-phase cutting stock problem.

Each processing operation has a high cost which implies items to be grouped before doing it, where

each group corresponds to one shelf. In the example above, we can consider three different classes and six different items. The size of the raw roll material corresponds to the size of the knapsack and the size of the shelf divisor corresponds to the loss generated by the first cutting phase. The values of δ and Δ are the minimum and maximum allowable width of the primary rolls. The value of an item can be its sale value.

Recently, this problem was considered by Hoto et al. [7] and Marques and Arenales [3], where exact and heuristic algorithms for the problem are presented. In [8], Hoto et al. considered the cutting stock version of the problem where a demand of items must be attended by the minimum number of bins. They use a column generation strategy, where the generation of each column reduces to the class constraint shelf knapsack problem.

Given an algorithm A_ε, for some $\varepsilon > 0$, and an instance I for some problem P we denote by $A_\varepsilon(I)$ the value of the solution returned by the algorithm A_ε when executed on instance I and by $\mathrm{OPT}(I)$ the value of an optimal solution for this instance. We say that A_ε is a polynomial time approximation scheme (PTAS) for a maximization problem if for any fixed $\varepsilon > 0$ and any instance I we have $A_\varepsilon(I) \geq (1-\varepsilon)\mathrm{OPT}(I)$. If the algorithm is also polynomial in $1/\varepsilon$ we say that A_ε is a fully polynomial time approximation scheme (FPTAS).

Results: In this paper we present approximation schemes for the SK and CCSK problems. We show that the CCSK problem cannot be approximated, unless $P = NP$, when each shelf must be a δ-shelf, but there is a fully polynomial time approximation scheme when the number of different items sizes in each class is bounded by a constant. To our knowledge, these are the first approximation results where shelves of non-null width are used in these problems.

Related Work: Knapsack problems are well studied problems under the approximation algorithms approach. In 1975, Ibarra and Kim [9] presented the first approximation scheme for the knapsack problem. In [1], Caprara et al. presented approximation schemes for two restricted versions of the knapsack problem, denoted by κKP and E-κKP. The κKP is the knapsack problem where the number of items chosen in the solution is at most k and the E-κKP is the problem where the number of items chosen in the solution is exactly k. Chekuri and Khanna [2] presented a PTAS for the Multiple Knapsack Problem (MKP). In this problem we have a set B of bins, each bin $j \in B$ with capacity b_j, and a set S of items, each item i with size s_i and value v_i. The objective is to find a subset $U \subseteq S$ of maximum total value such that U has a feasible packing in B. In [10], Shachnai and Tamir presented polynomial time approximation schemes for the class-constrained multiple knapsack (CCMK) problem. The CCMK problem consists in: given a set of items S, each item with value,

class and size and a set B of m bins, each bin $j \in B$ with capacity b_j and an upper bound C_j on the number of different classes to hold. The objective is to find a subset of S of maximum total value partitioned into m parts $S_1, \ldots, S_m$, each part S_j with total size at most b_j and at most C_j classes. In [12], Xavier and Miyazawa presented approximation algorithms for the class constrained shelf bin packing problem. In this problem, one has to pack a set of items in the minimum number of bins such that items are divided into shelves inside the bins, and each shelf contains only items of a same class.

Organization: In Section 2 we present a PTAS for the SK problem and in Section 3 we present a PTAS for the CCSK problem. In Section 4 we consider the CCSK problem when the shelves are δ-shelves. We present an inaproximability result and a fully polynomial time approximation scheme for the special case when the number of different sizes in each class is bounded by a constant. In Section 5, we present the concluding remarks.

2 A PTAS for the SK Problem

An instance I for the SK problem is a tuple (S, s, v, K, d, Δ) where S is a set of items, s and v are size and value functions over S, respectively; K is the size of the knapsack; d is the size of a shelf divisor, and Δ is an upper bound for the size of each shelf. All values are non-negatives. We denote by n the number of items in the set S.

Given a set of items $T \subseteq S$ of an instance for the SK problem, we denote by $s(T)$ the sum $\sum_{e \in T} s_e$ and by $v(T)$ the sum $\sum_{e \in T} v_e$.

A solution $(U, \mathcal{U})$ of an instance $I = (S, s, v, K, d, \Delta)$ for the problem SK is a set $U \subseteq S$ and a partition $\mathcal{U} = \{U_1, \ldots, U_k\}$ of U, where each set U_i is a shelf such that $s(U_i) \leq \Delta$ and the size $s(U, \mathcal{U}) := \sum_{i=1}^{k} (s(U_i) + d)$ of the solution is at most K. We say that $\mathcal{U}$ is a shelf packing of U. When there is no need to specify the partition of the items into shelves, a solution $(U, \mathcal{U})$ may be refereed only by the set U. Using this notation, we define the value of a solution U as the value $v(U)$.

The SK problem can be defined as follows: Given an instance I for the SK problem, find a solution $(U, \mathcal{U})$ of maximum value.

Given an instance $I = (S, s, v, K, d, \Delta)$ for the SK problem and a shelf packing $\mathcal{U}$ for $U \subseteq S$ that minimizes $s(U, \mathcal{U})$, we denote by $sp(U)$ this minimum size.

Some of the ideas used in the algorithm of this section have been proposed by Chekuri and Khanna [2]. Given an instance I for the SK problem and an optimum solution O^*, the algorithm performs

two main steps. First, the algorithm finds a set $U \subseteq S$ with $v(U) \geq (1 - O(\varepsilon))v(O^*)$ such that this set can be packed in a knapsack with shelves with size not greater than $sp(O^*)$. This is shown in the next subsection. In the second step, the algorithm obtains a solution $(U', \mathcal{U})$ with $U' \subseteq U$ and $v(U') \geq (1 - O(\varepsilon))v(U)$.

2.1 Finding the Items

The algorithm of this section, denoted by $Find_\varepsilon$, is presented in Figure 2. The algorithm generates a polynomial number of sets such that at least one has value very close to the optimal and its shelf packing size is not greater than the size of an optimum solution. First, the algorithm performs two reparameterization on the values of the items, steps 1–4. In the first reparameterization the algorithm obtains items values that are non-negative integer values bounded by $\lfloor n/\varepsilon \rfloor$, so that the value of any solution is bounded by $W = n\lfloor n/\varepsilon \rfloor$. This is obtained using the same rounding technique presented by Ibarra and Kim [9] for the knapsack problem. We define $M = \varepsilon V/n$ where V is the maximum value of an item in S. For each item $e \in S$ we define its value as $v'_e = \lfloor v_e/M \rfloor$. We denote by O' an optimal solution for the reparameterized instance. Using the same ideas of Ibarra and Kim it is not hard to see that $v(O') \geq (1 - \varepsilon)v(O^*)$. The key idea is the fact that the value of the solution O' is at most a factor nM smaller than the value of the solution O^*, due to the floor function in the rounding process. From now on, we only work with the reparameterized instance, since if we find a solution with value greater than or equal to $(1 - O(\varepsilon))v'(O')$ for the reparameterized instance we obtain a solution with value greater than or equal to $(1 - O(\varepsilon))v(O^*)$ for the original instance as the next lemma states.

Lemma 2.1 *Given an instance I for the* SK *problem, let I' be the instance obtained from I using the value function v' defined in steps 1–3 of the algorithm $Find_\varepsilon$. Let O^* and O' be optimum solutions of I and I', respectively. If U is a solution satisfying $v'(U) \geq (1-t\varepsilon)v'(O')$ then $v(U) \geq (1-(t+1)\varepsilon)v(O^*)$.*

Proof. Let U be a solution satisfying $v'(U) \geq (1 - t\varepsilon)v'(O')$ for some $t > 0$. From the definition of v' we can bound the value of $v(Q)$ for any set Q, as follows:

$$Mv'(Q) = M\sum_{e\in Q}\left\lfloor\frac{v_e}{M}\right\rfloor \leq \sum_{e\in Q} v_e = v(Q). \quad (1)$$

$$Mv'(Q) = M\sum_{e\in Q}\left\lfloor\frac{v_e}{M}\right\rfloor$$

$$\begin{aligned} &\geq M\sum_{e\in Q}\left(\frac{v_e}{M}-1\right)=\sum_{e\in Q}v_e - M|Q| \\ &\geq v(Q)-Mn = v(Q)-\varepsilon V. \end{aligned} \tag{2}$$

The proof of the lemma follows applying inequalities (1), (2) and the fact that $v(O^*)\geq V$.

$$\begin{aligned} v(U) &\geq Mv'(U)\geq(1-t\varepsilon)Mv'(O') \\ &\geq (1-t\varepsilon)Mv'(O^*)\geq(1-t\varepsilon)(v(O^*)-\varepsilon V) \\ &\geq (1-t\varepsilon)(v(O^*)-\varepsilon v(O^*)) = (1-t\varepsilon)(1-\varepsilon)v(O^*) \\ &\geq (1-(t+1)\varepsilon)v(O^*). \end{aligned}$$

□

The second reparameterization is a rounding step (step 4) where each item value is rounded down to the nearest power of $(1+\varepsilon)$. From now on, we denote the value function after these steps by v''. Notice that after this rounding step, for each item $e\in S$ we have $v''_e\geq(1-\varepsilon)v'_e$ and there are $h+1=\lfloor\log_{(1+\varepsilon)}\frac{n}{\varepsilon}\rfloor+1$ different values of items. The items are grouped by their values in sets $S_0,\ldots,S_h$, such that items in the same set have the same value (steps 4–7).

In step 8, the algorithm performs an enumeration of all possible tuples $(k_0,\ldots,k_h)$ where $k_i\in[0,\lceil h/\varepsilon\rceil]$ and $\sum_{i=0}^{h}k_i\leq h/\varepsilon$. These tuples are used to find sets $U_i\subseteq S_i$, $0\leq i\leq h$ such that $v(U_0\cup\ldots\cup U_h)\geq(1-O(\varepsilon))v'(O')$. This is explained later.

In what follows, we show that the set of all possible tuples satisfying such conditions can be obtained in polynomial time.

Lemma 2.2 *Let f be the number of g-tuples of non-negative integer values such that the sum of the values of the tuple is d. Then $f=\binom{d+g-1}{g}$.*

Lemma 2.3 *The number of different h-tuples $(k_0,\ldots,k_h)$ such that $\sum_{i=0}^{h}k_i=\frac{h}{\varepsilon}$ is $O(\frac{n}{\varepsilon}^{O(1/\varepsilon)})$.*

Proof. Let $d=\frac{h}{\varepsilon}$ and $g=h$. From Lemma 2.2, the number of possibilities to test, such that $\sum_i k_i=\frac{h}{\varepsilon}$, is

$$f=\binom{d+g-1}{g}\leq\binom{d+g}{\frac{d+g}{2}}\leq e2^{d+g}\leq O(2^{\frac{2}{\varepsilon}h})\leq O(\frac{n}{\varepsilon}^{O(1/\varepsilon)}). \tag{3}$$

□

Since the upper bound of Lemma 2.3 is also an upper bound for the number of tuples such that $\sum_i k_i=t$, for each $t\in\{1,\ldots,\frac{h}{\varepsilon}\}$, we can conclude that the number of tuples where $\sum_i k_i\leq\frac{h}{\varepsilon}$ can be computed in polynomial time.

ALGORITHM $Find_\varepsilon(I)$

Input: Instance $I = (S, s, v, K, \Delta)$ and a parameter $\varepsilon > 0$.

Output: A multset $\mathcal{Q}$, such that there is at least one set $U \in \mathcal{Q}$ with value close to the optimal.

1. Let $n \leftarrow |S|$, $V \leftarrow \max\{v_e : e \in S\}$, $M \leftarrow \frac{\varepsilon V}{n}$, and $h \leftarrow \lfloor \log_{1+\varepsilon} \frac{n}{\varepsilon} \rfloor$.

2. For each $e \in S$ do

3. $\quad v'_e \leftarrow \lfloor v_e / M \rfloor$

4. $\quad v''_e \leftarrow (1+\varepsilon)^k$ where $(1+\varepsilon)^k \leq v'_e < (1+\varepsilon)^{k+1}$.

5. For each $i \in \{0, \ldots, h\}$ do

6. $\quad$ let S_i be the set of items with value $(1+\varepsilon)^i$ in S

7. $\quad$ let $(e^i_1, \ldots, e^i_{n_i})$ be the items in S_i sorted in non-decreasing order of size.

8. Let $\mathcal{Q} \leftarrow \emptyset$ and T be the set of all possible tuples $(k_0, \ldots, k_h)$ such that $k_i \in \{0, \ldots, \frac{h}{\varepsilon}\}$ for $0 \leq i \leq h$ and $\sum_{i=0}^{h} k_i \leq \lceil \frac{h}{\varepsilon} \rceil$.

9. For each integral value w in the interval $[0, n\lfloor n/\varepsilon \rfloor]$ do

10. $\quad$ for each tuple $(k_0, \ldots, k_h)$ in T do

11. $\quad\quad$ for each $i \in \{0, \ldots, h\}$ do

12. $\quad\quad\quad$ let $U_i = \{e^i_1, \ldots, e^i_j\}$ such that $v''(U_i - e^i_j) < k_i(\frac{\varepsilon w}{h}) \leq v''(U_i)$

13. $\quad\quad$ $U \leftarrow (U_0 \cup \ldots \cup U_h)$

14. $\quad\quad$ $\mathcal{Q} \leftarrow \mathcal{Q} + U$.

15. Return $\mathcal{Q}$.

Figure 2: Algorithm to find sets with value close to $v'(O')$.

The algorithm generates sets U such that at least one has value $v(U) \geq (1 - O(\varepsilon))v'(O')$. Notice that we do not know in advance the value $v'(O')$. The algorithm tries each possible value of $v'(O')$ in the interval $[0, \ldots, n\lfloor n/\varepsilon \rfloor]$ (step 9).

Consider the sets $O'_i = O' \cap S_i, i = 0, \ldots, h$. The idea of steps 9–14 of the algorithm is to obtain subsets U_i with values very close to $v''(O'_i)$ using the tuples $(k_0, \ldots, k_h) \in T$ (T is the set of all valid tuples) generated in step 8. There is a tuple with integer values $k_i \in \{0, \ldots, \frac{h}{\varepsilon}\}$ such that $k_i \frac{\varepsilon v'(O')}{h} \leq v''(O'_i) < (k_i + 1)\frac{\varepsilon v'(O')}{h}$ as the next lemma states.

Lemma 2.4 *If O' is an optimal solution using v' function, then there exists a tuple $(k_0, \ldots, k_h) \in T$ such that for each i we have $k_i \in \{0, \ldots, \frac{h}{\varepsilon}\}$ and*

(i) $k_i \frac{\varepsilon v'(O')}{h} \leq v''(O'_i) < (k_i+1)\frac{\varepsilon v'(O')}{h}$,

(ii) $\sum_{i=1}^{h} k_i \frac{\varepsilon v'(O')}{h} \geq (1-3\varepsilon)v'(O')$ *and*

(iii) $\sum_{i=1}^{h} k_i \leq h/\varepsilon$.

Proof. Let $O' = O'_0 \cup \ldots \cup O'_h$ where $O'_i = S_i \cap O'$, for $0 \leq i \leq h$. For each i, let $k_i = \left\lfloor \frac{v''(O'_i)h}{\varepsilon v'(O')} \right\rfloor$. To prove item *(i)*, we use basic facts from the floor function to bound $v''(O'_i)$.

$$\begin{aligned} k_i \frac{\varepsilon v'(O')}{h} &= \left\lfloor \frac{v''(O'_i)h}{\varepsilon v'(O')} \right\rfloor \frac{\varepsilon v'(O')}{h} \\ &> \left(\frac{v''(O'_i)h}{\varepsilon v'(O')} - 1 \right) \frac{\varepsilon v'(O')}{h} \\ &= v''(O'_i) - \frac{\varepsilon v'(O')}{h}. \end{aligned}$$

Therefore, $v''(O'_i) < (k_i+1)\frac{\varepsilon v'(O')}{h}$. Also note that

$$\begin{aligned} k_i \frac{\varepsilon v'(O')}{h} &= \left\lfloor \frac{v''(O'_i)h}{\varepsilon v'(O')} \right\rfloor \frac{\varepsilon v'(O')}{h} \\ &\leq \frac{v''(O'_i)h}{\varepsilon v'(O')} \frac{\varepsilon v'(O')}{h} \\ &= v''(O'_i). \end{aligned}$$

The proof of the item *(ii)* is the following:

$$\begin{aligned} \sum_{i=0}^{h} k_i \frac{\varepsilon v'(O')}{h} &\geq \sum_{i=0}^{h} \left(v''(O'_i) - \frac{\varepsilon v'(O')}{h} \right) \\ &= v''(O') - 2\varepsilon v'(O') \\ &\geq \frac{v'(O')}{1+\varepsilon} - 2\varepsilon v'(O') \\ &\geq (1-3\varepsilon)v'(O'). \end{aligned}$$

To prove item *(iii)*, note that if $\sum_{i=0}^{h} k_i > h/\varepsilon$, then we would obtain a contradiction:

$$\begin{aligned} v'(O') &\geq v''(O') \geq \sum_{i=0}^{h} v''(O'_i) \\ &\geq \sum_{i=0}^{h} k_i \frac{\varepsilon v'(O')}{h} > v'(O'). \end{aligned}$$

□

Now we show how the algorithm obtains the items. Given a set S_i and a value k_i, the algorithm takes items in non-decreasing order of size until the total value v'' of the items becomes greater than or equal to $k_i\frac{\varepsilon v'(O')}{h}$ (steps 10–14). All possible sets are added to $\mathcal{Q}$, that is returned by the algorithm $Find_\varepsilon$.

The next lemma states that at least one of the sets obtained in this way has value very close to the optimal $v'(O')$ and has shelf packing size at most $sp(O')$.

Lemma 2.5 *If O' is an optimum solution for an instance (S, s, v', K, d, Δ) then there exists a set $U \in \mathcal{Q}$ obtained by the algorithm $Find_\varepsilon$ such that $v'(U) \geq (1-3\varepsilon)v'(O')$ and $sp(U) \leq sp(O')$.*

Proof.

Let $(k_0, \ldots, k_h)$ be a tuple satisfying Lemma 2.4. For each set S_i, the algorithm takes items in non-decreasing order of size until it obtains a set U_i with total value that is greater than or equal to $k_i\frac{\varepsilon v'(O')}{h}$. Since U_i and O'_i are subsets of S_i and all items of S_i have the same value, we have that $k_i\frac{\varepsilon v'(O')}{h} \leq v''(U_i) \leq v''(O'_i)$. The algorithm never takes more items than the set O'_i, i.e., $|U_i| \leq |O'_i|$ and the items of U_i are taken in non-decreasing order of size. Therefore, we conclude that $sp(U) \leq sp(O')$, i.e, the number of shelves needed to pack the set U is not greater than the number of shelves needed to pack the set O', and the total size of shelves used to pack U is not greater than the total size of shelves used to pack O'.

From Lemma 2.4 we have

$$\begin{aligned} v'(U) &\geq v''(U) = \sum_{i=0}^{h} v''(U_i) \\ &\geq \sum_{i=0}^{h} k_i \frac{\varepsilon v'(O')}{h} \\ &\geq (1-3\varepsilon)v'(O'). \end{aligned}$$

□

At last, the algorithm generates a polynomial number of sets, at least one with value $(1 - O(\varepsilon))v'(O')$ that can be packed optimally. In the next section we show how these sets are packed.

2.2 Packing the Items.

In the previous section we have obtained at least one set U of items such that its total value is very close to $v'(O')$ and $sp(U) \leq K$. Now we present an algorithm to obtain a solution $(U', \mathcal{U})$ with $U' \subseteq U$

such that $v'(U') \geq (1 - O(\varepsilon))v'(O')$. To obtain a shelf packing of U', we use the algorithm of Chekuri and Khanna for the multiple knapsack problem [2], which we denote by A_{CK}. We assume that the input of algorithm A_{CK} is a set U of items, a value Δ which is the size of the knapsacks and a value j which is the number of knapsacks. The algorithm returns a subset $U' \subseteq U$ partitioned in subsets $U' = U'_1 \cup \ldots \cup U'_j$ such that for each $i : 1 \leq i \leq j$, $s(U'_i) \leq \Delta$. The algorithm for packing the set U is given in the Figure 3.

ALGORITHM $Pack_\varepsilon(U)$

Input: set of items U each item $e \in U$ with size s_e and value v_e.

Output: Packing of $U' \subseteq U$ with $v'(U') \geq (1 - O(\varepsilon))v'(O')$ and size at most K.

Subroutine: A_{CK} (algorithm to pack the items into bins).

1. let $U' \leftarrow \emptyset$ and $V \leftarrow 0$

2. for $j = 1$ to n do

3. let $\mathcal{P} = P_1 \cup \ldots \cup P_j$ be the solution returned by the execution of $A_{CK}(U, \Delta, j)$.

4. let $S \leftarrow \sum_{i=1}^{j}(s(P_i) + d)$

5. if $S \leq K$ and $V \leq v'(\mathcal{P})$ then

6. let $U' \leftarrow \mathcal{P}$ and $V \leftarrow v'(\mathcal{P})$

7. return U'.

Figure 3: Algorithm to obtain a packing with almost optimum value.

Lemma 2.6 *If $U' = U'_1 \cup \ldots \cup U'_j$ is the packing generated by the algorithm $Pack_\varepsilon$ with items $U' \subseteq U$ then $v'(U') \geq (1 - \varepsilon)v'(U)$ and $\sum_{i=1}^{j}(s(U'_i) + d) \leq K$.*

Proof. Let $\mathcal{P}^* = P^*_1 \cup \ldots \cup P^*_q$ be an optimal shelf packing for an optimal solution O'. From the proof of Lemma 2.5 it is easy to construct an injection $f : U \rightarrow O'$ where $s(e) \leq s(f(e))$ for each item $e \in U$. That is, we can construct a shelf packing for the set U using the partition of $\mathcal{P}^*$. When the algorithm $Pack_\varepsilon$ performs the call $\mathrm{A}_{CK}(U, \Delta, q)$, we obtain a solution $\mathcal{P} = P_1 \cup \ldots \cup P_q$ such that $v'(\mathcal{P}) \geq (1 - \varepsilon)v'(U)$, since A_{CK} is a PTAS, and $\sum_{i=1}^{q}(P_i + d) \leq \sum_{i=1}^{q}(P^*_i + d)$, since an injection f exists. The algorithm $Pack_\varepsilon$ returns a solution such that $v'(U') \geq v'(\mathcal{P})$, and therefore the lemma follows. □

2.3 The Algorithm

The PTAS for the problem SK is presented in Figure 4. For each value in the interval $[0, n\lfloor\frac{n}{\varepsilon}\rfloor]$, the algorithm $Find_\varepsilon$ generates a polynomial number of sets U, at least one of the sets with value very close to the value of an optimal solution O' and has a packing of size at most the size of the optimal. For all possibilities of U it uses the algorithm $Pack_\varepsilon$ to pack these sets. The solution returned by the algorithm is the packing of a set U' with maximum value satisfying the condition that its packing size is not greater than the capacity of the knapsack.

ALGORITHM $ASK_\varepsilon(I)$

Input: Instance $I = (S, s, v, K, \Delta, d)$.

Output: A solution U' of I with $v'(U') \geq (1 - O(\varepsilon))v'(O')$.

1. Let $\mathcal{Q} \leftarrow Find_\varepsilon(I)$.

2. Let $U' \leftarrow \emptyset$ and $v'(U') \leftarrow 0$.

3. For each $U \in \mathcal{Q}$ do

4. $\quad Q \leftarrow Pack_\varepsilon(U)$

5. $\quad$ if $v'(Q) > v'(U')$ then $U' \leftarrow Q$.

6. Return U'.

Figure 4: Approximation Scheme for the problem SK.

The time complexity of algorithm $Find_\varepsilon$ is dominated by the time to execute steps 9–14, which is $M = O(n^2(\log_{1+\varepsilon}\frac{n}{\varepsilon})O(\frac{n}{\varepsilon}^{O(1/\varepsilon)})$. Therefore, the number of sets returned by algorithm $Find_\varepsilon$ is also bounded by M. The time complexity of algorithm $Pack_\varepsilon$ is $O(nT_{CK}(n, \epsilon))$ where $T_{CK}(n, \epsilon)$ is the time complexity of the PTAS A_{CK}, presented by Chekuri and Khanna [2]. We can conclude with the following theorem.

Theorem 2.7 *The algorithm ASK_ε is a PTAS for the* SK *problem.*

3 Approximation Scheme for the CCSK Problem

Now we consider the class constrained version of the SK problem, which we denote by CCSK. We assume the same notation used for the SK problem.

In this case, an instance I for the CCSK problem is a tuple $(S, s, v, K, d, \Delta, c)$ where c is a class function over S.

A solution $(U,\mathcal{U})$ of an instance I for the problem CCSK is a set $U \subseteq S$ and a partition $\mathcal{U} = \{U_1, \ldots, U_k\}$ of U, where each set U_i is a shelf such that $s(U_i) \leq \Delta$ and all items in U_i have the same class. Two subsequent shelves must be separated by a shelf divisor and the total size of the solution must be at most K. The goal is to obtain a solution of maximum value. We present an approximation scheme for the problem CCSK given that there is an algorithm that solve another problem we call *Small*.

Problem Small: Given an instance I for the CCSK problem, where all items are of the same class and given a value w, find a solution $(U,\mathcal{U})$ of I with value w and smallest size, if one exists.

We say that an algorithm SS_ε is *ε-relaxed* for the problem Small if given an instance I and a value w, the algorithm generates a solution $(U,\mathcal{U})$ with all items of the same class with $(1-O(\varepsilon))w \leq v(U) \leq w$ and $sp(U) \leq sp(O)$, where O is a solution with value w and smallest size. Such solution $(U,\mathcal{U})$ is called an *ε-relaxed* solution.

It is not hard to see that we can use the same ideas of the algorithm ASK_ε to obtain an ε-relaxed algorithm for the problem Small. Given a set of items of class j and a value w the algorithm generates a polynomial number of sets such that at least one has value very close to w and its packing size is smaller than the packing of an optimal set. The algorithm returns the smallest packing such that its value is at least $(1 - O(\varepsilon))w$ and at most w. If none exists, then the minimum size of a solution is ∞, since in this case no solution with value w exists. The following lemma is valid.

Lemma 3.1 *There exists an ε-relaxed algorithm for problem* Small.

In Figure 5 we present an approximation scheme for CCSK using a subroutine for the problem Small.

In steps 1–3 the original instance is reparameterized in such a way the item values are non-negative integer values bounded by $\lfloor n/\varepsilon \rfloor$. Therefore, the value of any solution is bounded by $W = O(n^2/\varepsilon)$. This leads to a polynomial time algorithm using a dynamic programming approach with only $O(\varepsilon)$ loss on the total value of the solution found by the algorithm.

In steps 4–8, the algorithm generates ε-relaxed solutions for each problem Small obtained from the reparameterized instances of each class and each possible value w. The solutions are stored in variables $A_{j,w}$, for each class j and each possible value w.

In steps 9–15 problem CCSK is solved using dynamic programming. There is a table $T_{j,w}$ indexed by classes j and all possible values w. It stores the smaller solution using items of classes $\{1, \ldots, j\}$ that has value w. The basic idea is to solve the following recurrence:

ALGORITHM $G_\varepsilon(I)$ where $I = (S, c, s, v, K, d, \Delta)$

Subroutine: SS_ε (ε-relaxed algorithm for problem SMALL).

1. *% reparameterize the instance by value*

2. Let $n \leftarrow |S|$, $V \leftarrow \max\{v_e : e \in S\}$, $M \leftarrow \varepsilon V/n$ and $W \leftarrow n\lfloor n/\varepsilon \rfloor$.

3. For each item $e \in S$ do $v'_e \leftarrow \left\lfloor \frac{v_e}{M} \right\rfloor$

4. *% generate an ε-relaxed solution for each class*

5. For class $j \leftarrow 1$ to m do

6. for value $w \leftarrow 1$ to W do

7. let S_j be the set of items in S with class j

8. $A_{j,w} \leftarrow SS_\varepsilon(S_j, s, v', K, d, \Delta, w)$.

9. *% Find a solution with classes $\{1, \ldots, j\}$ for each possible value w.*

10. For class 1 do

11. for value $w \leftarrow 1$ to W do

12. $T_{1,w} \leftarrow A_{1,w}$.

13. For class $j \leftarrow 2$ to m do

14. for value $w \leftarrow 1$ to W do

15. $T_{j,w} \leftarrow$(solution in $\{T_{j-1,w}, A_{j,w}, \min_{1 \le k < w}\{T_{j-1,k} + A_{j,w-k}\}\}$ of value in $[(1-\varepsilon)w, w]$ and minimum size).

16. Let U be the solution $T_{m,w}$, $1 \le w \le V$ with maximum value w and size $s(U) \le K$.

17. Return U.

Figure 5: Generic algorithm for CCSK using subroutine for problem SMALL.

$$T_{j,w} := \min\{T_{j-1,w}, A_{j,w}, \min_{1 \le k < w}\{T_{j-1,k} + A_{j,w-k}\}\}.$$

Finally, given that there are m classes, in steps 16–17 a solution generated with maximum value w is returned.

To prove that G_ε is an approximation scheme we consider that algorithm SS_ε, used as subroutine, is an ε-relaxed algorithm for the problem SMALL.

Lemma 3.2 *If algorithm G_ε uses an ε-relaxed algorithm as subroutine and if $O_{j,w}$ is a solution using classes $\{1, \ldots, j\}$, with $w := v'(O_{j,w})$ and minimum size, then $T_{j,w}$ exists and $v'(T_{j,w}) \ge (1-\varepsilon)w$ and*

$s(T_{j,w}) \leq s(O_{j,w})$.

Proof. We can prove this fact by induction on the number of classes. The base case consider only items with class 1 and can be proved from the fact that subroutine SS_ε is an ε-relaxed algorithm (that is, $T_{1,w} = A_{1,w}$).

Consider a solution $O_{j,w}$ with value $w := v'(O_{j,w})$ using items of classes $\{1, \ldots, j\}$.

If $O_{j,w}$ uses only items of class j, then we have a solution $A_{j,w}$ which is obtained from subroutine SS_ε, which by assumption is an ε-relaxed algorithm. Therefore, $v'(A_{j,w}) \geq (1-\varepsilon)v'(O_{j,w})$ and $s(A_{j,w}) \leq s(O_{j,w})$.

If $O_{j,w}$ uses only items of classes $1, \ldots, j-1$, by induction, $T_{j-1,w}$ exists and $v'(T_{j-1,w}) \geq (1-\varepsilon)v'(O_{j,w})$ and $s(T_{j-1,w}) \leq s(O_{j,w})$.

If $O_{j,w}$ uses items of class j and items of other classes, denote by O_1 and O_2 two solutions obtained partitioning $O_{j,w}$ such that O_1 contains the items of class different than j and O_2 contains the items of class j. Let $k := v'(O_1)$. By induction, there are solutions $T_{j-1,k}$ and $A_{j,w-k}$ such that

$$v'(T_{j-1,k}) + v'(A_{j,w-k}) \geq (1-\varepsilon)k + (1-\varepsilon)(w-k) = (1-\varepsilon)v'(O_{j,w}) \quad \text{and}$$

$$s(T_{j-1,k}) + s(A_{j,w-k}) \leq s(O_1) + s(O_2) = s(O_{j,w}).$$

□

Theorem 3.3 *If I is an instance for the problem* CCSK *and SS_ε is an ε-relaxed polynomial time algorithm for the problem* SMALL *then the algorithm G_ε with subroutine SS_ε is a polynomial time approximation scheme for* CCSK. *Moreover, if SS_ε is also polynomial time in $1/\varepsilon$, the algorithm G_ε is a fully polynomial time approximation scheme.*

Proof. Let O be an optimum solution for instance I. Let $T_{m,w}$ be a solution found by the algorithm (remember that it uses rounded items).

$$\begin{aligned}
v(T_{m,w}) &= \sum_{e \in T_{m,w}} v_e \geq \sum_{e \in T_{m,w}} v'_e M = M v'(T_{m,w}) \\
&\geq M(1-\varepsilon)w \geq M(1-\varepsilon)\sum_{e \in O} v'_e \\
&\geq M(1-\varepsilon)\sum_{e \in O}(\frac{v_e}{M} - 1) \geq M(1-\varepsilon)(\sum_{e \in O}\frac{v_e}{M} - n) \\
&= (1-\varepsilon)(\sum_{e \in O} v_e - nM) = (1-\varepsilon)(\mathrm{OPT} - \varepsilon V) \\
&\geq (1-\varepsilon)(\mathrm{OPT} - \varepsilon \mathrm{OPT}) \\
&\geq (1-2\varepsilon)\mathrm{OPT}.
\end{aligned}$$

Since the solution returned by the algorithm G_ε has value $v(T_{m,w})$, we have that $G_\varepsilon(I) \geq (1-2\varepsilon)\mathrm{OPT}$.

Let m and n be the number of classes and the number of items, respectively. Since $W = n\lfloor n/\varepsilon \rfloor$, the time complexity of steps 5–8 of algorithm G_ε, is $O(mn^2/\varepsilon \cdot T_{SS}(n,\epsilon))$, where $T_{SS}(n,\epsilon)$ is the time complexity of subroutine SS_ε. For a given class j and value w the solution $T_{j,w}$, in step 15, can be computed in time $O(w)$, and the time complexity of steps 13–15 is $O(mn^4/\varepsilon^2)$. The overall time complexity of the algorithm G_ε is $O(mn^4/\varepsilon^2 + mn^2/\varepsilon \cdot T_{SS}(n,\epsilon))$. Therefore, if SS_ε has polynomial time in n (and in $1/\varepsilon$) then algorithm G_ε is a (fully) polynomial time approximation scheme. □

From Lemma 3.1 and Theorem 3.3, we can conclude the following theorem.

Theorem 3.4 *There exists a PTAS for the* CCSK *problem.*

4 Approximation Results for the SK Problem with δ-shelves

In this section, we consider the SK problem when each shelf must have size in $[\delta, \Delta]$. As mentioned before, this case has applications in the iron and steel industry [5].

We first prove an inapproximability result for the SK problem with *δ-shelves*. This result also extends to the CCSK problem with *δ-shelves*. Furthermore, we present a FPTAS for the case when the number of different items size in each class is bounded by a constant k.

4.1 Inapproximability of the problem SK with δ-shelves

We present a gap-introducing reduction to prove the inapproximability of this case (see [11]). The proof is made by reducing the partition problem to the SK problem.

Lemma 4.1 *There is a gap-introducing reduction transforming instance I_1 of the Partition Problem (PP) to an instance I_2 of the* SK *problem such that:*

- *If there exists a partition for instance I_1 then $OPT(I_2) = \delta$, and*
- *if there is no partition for instance I_1 then $OPT(I_2) = 0$.*

Proof. Let $I_1 = (S, s)$ be the instance of Partition Problem where each item $e \in S$ has size s_e. We construct an instance I_2 such that $OPT(I_2) \in \{0, \delta\}$. Let $I_2 = (S, s', v, K, 0, \Delta, \delta)$ be an instance for the SK problem obtained from I_1 as follows: For each item $e \in S$ we have $v_e = s_e\alpha$ and $s'_e = s_e\alpha$,

where α is an integer constant. Clearly, there exists a partition for instance (S, s) if and only if there exists a partition of instance (S, s'). Let $\delta = \frac{\sum_{e \in S} s'_e}{2}$, $\Delta = \delta + (\alpha - 1)$ and $K = \Delta$. Notice that the size of a shelf divisor is zero.

First note that any size s'_e is a multiple of α and therefore, any solution of I_2 is also multiple of α. Since δ is multiple of α, we have $\delta < \Delta < \delta + \alpha$ and we conclude that there is no solution to instance I_2 with size greater than δ.

If instance I_1 can be partitioned, then the optimal solution of instance I_2 has value δ and the knapsack is filled until δ. If instance I_2 can not be partitioned, then the only solution with size multiple of α that respects the limits δ and Δ has value 0 and it packs no items.

□

Theorem 4.2 *There is no r-approximation algorithm for the problem* SK *with δ-shelves when $0 < \delta \leq \Delta$, for any $r > 0$, unless $P = NP$.*

4.2 Approximation Scheme for a special case of the problem CCSK with δ-shelves

In this section we consider a special case of the problem CCSK with δ-shelves, where the number of different items sizes for each class is bounded by a constant k. As we show in the next theorem, this special case is NP-hard.

Theorem 4.3 *The problem restricted to instances with at most a constant k of different sizes in each class is still NP-hard.*

Proof. The theorem is valid since the knapsack problem is a particular case when each item is of a different class and $\Delta = \infty$, $\delta = 0$ and $d = 0$. □

We present a fully polynomial time approximation scheme for this problem. The algorithm is the same as presented in the section 3. In this case, we only need to present an ε-relaxed algorithm to solve problem Small, used as subroutine by the algorithm G_ε, that is polynomial time both in the input size and in $1/\varepsilon$. In fact, we show that an algorithm for the problem Small does not need to compute solutions for every value w to obtain a fully polynomial time approximation scheme for the CCSK problem.

4.2.1 The k-Pack Problem

Before presenting the algorithm to solve problem Small, consider the problem, denoted by k-Pack, which consists in packing n one-dimensional items with at most a constant k of different items sizes into the minimum number of bins of size Δ, each bin filled by at least δ.

The algorithm to solve problem k-Pack uses a dynamic programming strategy combined with the generation of all configurations of packings of items into one bin. In Figure 6 we present the algorithm that generates a function B that returns the minimum number of bins to pack an input list, under the restrictions of the problem k-Pack. For our purposes, we also need that the function B returns the partition of the input list into bins. For simplicity, we let to the interested reader its conversion to an algorithm that also returns the partition of the input list into bins.

In step 3, the algorithm generates all possible subsets of items that can be packed into one bin. Notice that the number of different tuples is $O(n^k)$ and the algorithm just need to test each one of these tuples if they satisfy the properties of the bin, i.e, $\delta \leq \sum_{i=1}^{k} q_i s_i \leq \Delta$. The overall time complexity of these steps is $O(n^k)$. In each iteration of the while command, steps 5–10, the algorithm uses the knowledge of instances that uses i bins to compute instances that uses $i+1$ bins. Notice that each set Q_i can have at most $O(n^k)$ tuples. The time complexity of the entire algorithm is $O(n^{2k+1})$.

The following theorem is straightforward.

Theorem 4.4 *The algorithm P_k generates a function that returns the minimum number of bins to pack any sublist of the input list L of problem k-Pack. Moreover, the algorithm P_k has a polynomial time complexity.*

4.2.2 Solving Problem Small

The following lemma states the relationship of a solution for problem Small and the problem k-Pack.

Lemma 4.5 *If $O = (L, P)$ is an optimum solution of an instance $I = (S, s, v', K, d, \Delta, \delta, w)$ for the problem Small, $L \subseteq S$ and $P = (P_1, \ldots, P_j)$ then $j \geq B(L)$, where $B(L)$ is the minimum number of bins to pack L into bins of size Δ, filled by at least δ.*

Proof. Notice that items packed in the optimum solution O are separated by shelf divisors of size d and the size of a shelf is at least δ and at most Δ. Items in shelves can be considered as a packing into bins of size Δ occupied by at least δ. Since $B(L)$ is the minimum number of such bins to pack L, the number of shelves of L is at least $B(L)$. □

ALGORITHM $P_k(L, \delta, \Delta)$

1. Let $s_1, \ldots, s_k$ the k different sizes occurring in list L.

2. Let d_i be the number of items in L of size s_i, $i = 1, \ldots, k$.

3. Let Q_1 be the set of all tuples $(q_1, \ldots, q_k)$ such that $0 \leq q_i \leq d_i$, $i = 1, \ldots, k$ and $\delta \leq \sum_{i=1}^{k} q_i s_i \leq \Delta$.

4. let $i \leftarrow 1$

5. while $(d_1, \ldots, d_k) \notin Q_i$ do

6. $\quad Q_{i+1} \leftarrow \emptyset$

7. $\quad$ for each $q' \in Q_1$ and $q'' \in Q_i$ do

8. $\quad\quad q \leftarrow q' + q''$

9. $\quad\quad$ if $q \notin Q_i$ then $Q_{i+1} \leftarrow Q_{i+1} \cup \{q\}$

10. $\quad i \leftarrow i + 1$

11. let $B(q) \leftarrow j$ for all $q \in Q_j$, $1 \leq j \leq i$

12. return B

Figure 6: Algorithm to find the minimum number of bins to pack any subset of L.

Corollary 4.6 *If $O = (L, P)$ is an optimum solution of an instance $I = (S, s, v', K, d, \Delta, \delta, w)$ for the problem* SMALL*, then $sp(O) \geq s(L) + B(L)d$.*

In Figure 7, we present an algorithm for solving a relaxed version of the problem SMALL, which is sufficient to our purposes. The algorithm first considers all possible configurations of solutions for the problem SMALL, without considering the value of each item. This step is performed by a subroutine to solve problem k-PACK. Instead of finding each possible attribution of values for each configuration, the algorithm only generates valid configurations with maximum value. For a given value w, the algorithm only returns a solution if the value is a maximum value for some configuration. Notice that we return the smallest packing that has the given value.

Theorem 4.7 *If I is an instance for the* CCSK *problem with at most k different items sizes in each class and algorithm G_ε is executed with subroutine k-SS then $G_\varepsilon(I) \geq (1 - \varepsilon)\mathrm{OPT}(I)$.*

Proof. Consider an optimum solution O to the instance I with the function value v'. Let Q_c be the set of items of class c used in this optimal solution. This set of items corresponds to a configuration q_c that is packed optimally by Corollary 4.6. The algorithm *k-SS* returns items of maximum value

ALGORITHM k-$SS(S, s, v', K, d, \Delta, \delta, w)$

Subroutine: P_k (Subroutine to solve problem k-PACK).

1. Let $B \leftarrow P_k(S, \delta, \Delta)$.

2. Let $s_1, \ldots, s_k$ the k different sizes occurring in list S.

3. Let d_i be the number of items in S of size s_i, $i = 1, \ldots, k$.

4. Let Q be the set of all tuples $(q_1, \ldots, q_k)$ such that $0 \leq q_i \leq d_i$, $i = 1, \ldots, k$.

5. For each $q = (q_1, \ldots, q_k) \in Q$ do

6. let $P(q)$ the packing obtained using function $B(q)$ placing for each size s_j, q_j items of L with size s_j and greatest values.

7. Let $Q_w \leftarrow \{q \in Q : w = v(P(q))\}$.

8. If $Q_w \neq \emptyset$ then

9. return $q \in Q_w$ such that $s(q)$ is minimum

10. else

11. return $\emptyset$.

Figure 7: An ε-relaxed algorithm for δ-shelves.

corresponding to this configuration. If the algorithm k-SS does not return this optimal solution to G_ε, it finds another configuration with the same value but with smaller size. It follows from Theorem 3.3 that the optimal solution found by algorithm G_ε with the subroutine k-SS is a FPTAS to CCSK. □

5 Concluding Remarks

In this paper we have presented approximation schemes for shelf knapsack problems. These problems have many applications, and to the best of our knowledge, this is the first paper to present approximation results for them.

6 Acknowledgements

We would like to thank the anonymous referees that present helpful suggestions which improved the presentation of this paper.

References

[1] A. Caprara, H. Kellerer, U. Pferschy and D. Pisinger. Approximation algorithms for knapsack problems with cardinality constraints. *European Journal of Operational Research*, 123:333–345, 2000.

[2] C. Chekuri and S. Khanna. A ptas for the multiple knapsack problem. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 213–222, 2000.

[3] F. P. Marques and M. Arenales. The constrained compartmentalized knapsack problem. *To appear in Computer & Operations Research.*

[4] W. Fernandez de la Vega and G. S. Lueker. Bin packing can be solved within $1 + \epsilon$ in linear time. *Combinatorica*, 1(4):349–355, 1981.

[5] J. S. Ferreira, M. A. Neves, and P. Fonseca e Castro. A two-phase roll cutting problem. *European Journal of Operational Research*, 44:185–196, 1990.

[6] A. M. Frieze and M. R. B. Clarke. Approximation algorithms for the m-dimensional 0-1 knapsack problem: worst-case and probabilistic analysis. *European Journal of Operational Research*, 15:100–109, 1984.

[7] R. Hoto, N. Maculan, M. Arenales and F. P. Marques. Um novo procedimento para o cálculo de mochilas compartimentadas. *Investigação Operacional*, 22:213–234.

[8] R. Hoto, M. Arenales and N. Maculan. The one dimensional compartmentalized cutting stock problem: a case study. *To appear in European Journal of Operational Research.*

[9] O. H. Ibarra and C. E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *Journal of the Association for Computing Machinery*, 22:463–468, 1975.

[10] H. Shachnai and T. Tamir. Polynomial time approximation schemes for class-constrained packing problems. *Journal of Scheduling*, 4(6):313–338, 2001.

[11] V. Vazirani. *Approximation Algorithms.* Springer-Verlag, 2001.

[12] E. C. Xavier and F. K. Miyazawa. A one-dimensional bin packing problem with shelf divisions. *Second Brazilian Symposium on Graphs, Algorithms and Combinatorics. Electronic Notes in Discrete Mathematics*, 19:329–335, 2005.