Available online at www.sciencedirect.com

SCIENCE DIRECT®

Operations Research Letters 32 (2004) 5 – 14

Operations Research Letters

www.elsevier.com/locate/dsw

On the two-dimensional Knapsack Problem

Alberto Caprara*, Michele Monaci

Dipartimento di Elettronica, Informatica e Sistemistica, Università di Bologna, Viale Risorgimento, 2, I-40136, Bologna, Italy

Received 12 August 2002; received in revised form 28 February 2003; accepted 3 April 2003

Abstract

We address the two-dimensional Knapsack Problem (2KP), aimed at packing a maximum-profit subset of rectangles selected from a given set into another rectangle. We consider the natural relaxation of 2KP given by the one-dimensional KP with item weights equal to the rectangle areas, proving the worst-case performance of the associated upper bound, and present and compare computationally four exact algorithms based on the above relaxation, showing their effectiveness.
© 2003 Elsevier B.V. All rights reserved.

Keywords: Two-dimensional Knapsack; Upper bound; Approximation Algorithm; Worst-case Performance; Exact Algorithm

1. Introduction

We address the (*orthogonal*) *two-dimensional Knapsack Problem* (2KP), whose input consists of a set $N = \{1, \ldots, n\}$ of "small" rectangles (*items*), the jth having a *width* w_j, a *height* h_j and a *profit* p_j, and a "large" rectangle (*knapsack*) of width W and height H. The objective is to pack a maximum-profit subset of the items into the knapsack, with the constraint that items do not overlap and each of them, say the jth, must have its edge of height h_j parallel to the edge of the knapsack of height H (such a requirement is called *orthogonal packing without rotation*). 2KP is a generalization of the famous *one-dimensional Knapsack Problem* (KP), arising as a special case if $h_j = H$ for all $j \in N$. Whereas KP is solvable in pseudopolynomial time by dynamic programming, 2KP is strongly NP-hard, since in the special case in which all item heights are equal, the problem of testing whether the whole set N of items fits in the knapsack is equivalent to the well-known (one-dimensional) bin packing problem.

Methods for the practical solution of 2KP have widely been studied in the literature since the seminal work of Gilmore and Gomory [14]. In many practical applications, additional constraints are imposed on the problem, such as *guillotine* or *k-stage* solution patterns. For a comprehensive annotated bibliography on 2KP and variations, see [8]. On the other hand, no approximation algorithms with worst-case performance guarantee appear to be known for 2KP, which is surprising since a few results in this direction are known for related two-dimensional packing problems, see e.g. [6]. To the best of our knowledge, there is no result that rules out the existence of polynomial time approximation schemes for 2KP.

In this paper, we mainly focus on upper bounds and exact algorithms for 2KP. Previous works in this direction are those by Beasley [2], Hadjiconstantinou and Christofides [16], Boschetti et al. [4] and Fekete and Schepers [9,13]. The first three references present integer linear programming formulations of

* Corresponding author.
E-mail addresses: acaprara@deis.unibo.it (A. Caprara), mmonaci@deis.unibo.it (M. Monaci).

0167-6377/03/$ - see front matter © 2003 Elsevier B.V. All rights reserved.
doi:10.1016/S0167-6377(03)00057-9

the problem which are used to derive upper bounds through Lagrangian or surrogate relaxation. Moreover, the first two references illustrate branch-and-bound algorithms based on these upper bounds, in which branching selects at the same time an item to pack in the knapsack and the position in which it is packed, whereas the last one presents heuristic algorithms that are based on the relaxations and are often capable of finding provably optimal solutions. The last reference presents upper bounding procedures based on the so-called *dual feasible functions* (see Fekete and Schepers [10,13]), which lead to the solution of a suitable KP instance. This upper bound is embedded into a branch-and-bound algorithm in which the branching rule defines the set S of items to pack in the knapsack *without* specifying the corresponding position. Each time a new item is added to S, the algorithm checks whether the items in S fit in the knapsack by a separate enumerative procedure. (More details about this algorithm will be given in Section 3.) A branching scheme that specifies that a given item will be packed, without fixing the position in which it will be packed, has also been proposed by Martello and Vigo [20] to solve the two-dimensional bin packing problem (2BPP). In practice, the approach in [13] widely outperforms the previous ones.

In Section 2, we consider the natural relaxation of 2KP given by the (one-dimensional) KP with item weights equal to the rectangle areas, showing that the associated optimal value is at most 3 times the 2KP solution value, and that the worst-case ratio of 3 is tight. As a byproduct, we get a polynomial $(\frac{1}{3}-\varepsilon)$-approximation algorithm for 2KP (for any $\varepsilon > 0$), which appears to be the first nontrivial approximation algorithm for the problem. We also note that a completely analogous result holds for the two-dimensional Bin Packing Problem and its one-dimensional relaxation. In Section 3, we present four enumerative schemes for the optimal solution of 2KP, all based on the above relaxation, and in which the branching step fixes the set of items packed in the solution without specifying the packing layout. The feasibility of the current set of items, i.e. the fact that they fit into the knapsack, is tested by using an exact algorithm for the two-dimensional Strip Packing Problem as a black box. The difference among the various enumerative schemes is the frequency of this feasibility test. Computational comparison in Section 4 shows that different policies lead to substantially different running times, and that the best of our enumerative schemes turns out to be competitive with the algorithm in [13].

2. A natural one-dimensional relaxation

A natural relaxation of 2KP (see e.g. [4]) is the following KP:

$$\max \sum_{j\in N} p_j x_j, \tag{1}$$

$$\sum_{j\in N} (w_j h_j) x_j \leqslant WH, \tag{2}$$

$$x_j \in \{0,1\} \quad (j\in N), \tag{3}$$

in which each item j $(j\in N)$ has profit p_j and weight $w_j h_j$, equal to the area of rectangle j in the 2KP instance, and the knapsack capacity is WH, equal to the area of the knapsack in the 2KP instance. For a 2KP instance I, let $OPT(I)$ denote the optimal solution value and $U_{\mathrm{KP}}(I)$ the upper bound on $OPT(I)$ corresponding to the optimal solution value of (1)–(3). The (*absolute*) *worst-case performance ratio* of U_{KP} is defined as

$$r(U_{\mathrm{KP}}) := \inf_I \left\{ \frac{OPT(I)}{U_{\mathrm{KP}}(I)} \right\}. \tag{4}$$

We can easily determine $r(U_{\mathrm{KP}})$ starting from a nontrivial result of Steinberg [21].

Lemma 1 (Steinberg [21]). *Given a set S of items having width $w_j \leqslant \bar{W}$ and height $h_j \leqslant \bar{H}$ $(j\in S)$ such that*

$$2\sum_{j\in S} w_j h_j \leqslant \bar{W}\,\bar{H} - (2w_{\max}-\bar{W})^+(2h_{\max}-\bar{H})^+, \tag{5}$$

where $w_{\max} := \max_{j\in S} w_j$, $h_{\max} := \max_{j\in S} h_j$ and $x^+ := \max(0,x)$, it is possible to pack the items in S into a knapsack of width $\bar{W}$ and height $\bar{H}$ in polynomial time.

Lemma 2. *Given a set S of items having width $w_j \leqslant W$ and height $h_j \leqslant H$ $(j\in S)$ such that $\sum_{j\in S} w_j h_j \leqslant WH$, it is possible to pack the items in S into three knapsacks of width W and height H in polynomial time.*

```
Algorithm $\frac{1}{3}$-APX:
    Let $\varepsilon$ be the required accuracy and define $\bar{\varepsilon} = \varepsilon/3$.
(1) Find a $(1-\bar{\varepsilon})$-approximated solution for KP, (1)–(3) letting $S$ denote the set of items selected
    in the solution.
(2) Pack the items in $S$ into a knapsack of width $W$ and height $2H$.
(3) Define sets $J_1$, $J_2$ and $J_3$ according to (6)–(8).
(4) Pack into the knapsack (of width $W$ and height $H$) the item set $J^*$ with maximum profit among
    $J_1$, $J_2$ and $J_3$ according to Lemma 1.
```

Fig. 1. The $(\frac{1}{3}-\varepsilon)$-approximation algorithm for 2KP.

Proof. By Lemma 1, it is possible to pack all the items in S into a knapsack of width W and height $2H$, since inequality (5) trivially holds for such a knapsack (in particular $(2h_{\max} - 2H)^+ = 0$). Let (x_j, y_j) be the coordinates at which the bottom-left corner of item j is placed $(j \in S)$, imagining the edge of width W of the knapsack to be horizontal and its lower left corner to have coordinates $(0, 0)$. Partition the item set S in the following way:

$$J_1 := \{j \in S : y_j + h_j \leqslant H\}, \tag{6}$$

$$J_2 := \{j \in S : y_j \geqslant H\}, \tag{7}$$

$$J_3 := \{j \in S : y_j < H < y_j + h_j\}. \tag{8}$$

Since $h_j \leqslant H$ for $j \in S$, each of sets J_1, J_2, J_3 fits into a knapsack of width W and height H and the corresponding packing layout is given by the item coordinates. □

Proposition 1. $r(U_{\text{KP}}) = \frac{1}{3}$.

Proof. Let S be the set of items in the optimal solution of (1)–(3), of value U_{KP}. The best of the three 2KP solutions among J_1, J_2 and J_3 defined in the proof of Lemma 2 has value

$$\max\left\{\sum_{j\in J_1} p_j, \sum_{j\in J_2} p_j, \sum_{j\in J_3} p_j\right\} \geqslant U_{\text{KP}}(I)/3.$$

Tightness of the $\frac{1}{3}$ value is shown by the following instance I: $n = 3$, $w_j = W/\sqrt{3}$, $h_j = H/\sqrt{3}$, $p_j = 1$ $(j = 1, 2, 3)$. We have $U_{\text{KP}}(I) = 3$ (and $S = \{1, 2, 3\}$), whereas $OPT(I) = 1$ since at most one item can be packed. □

The proof of Proposition 1 immediately leads to a pseudopolynomial $\frac{1}{3}$-approximation algorithm for 2KP, i.e. an algorithm that produces a feasible solution of value no less than $OPT(I)/3$ for any instance I. This may be turned into a polynomial $(\frac{1}{3} - \varepsilon)$-approximation algorithm for 2KP for any $\varepsilon > 0$ (the running time also polynomial in $1/\varepsilon$) by using any fully polynomial approximation scheme for KP (see e.g. [17]). The algorithm is presented in Fig. 1. Letting S^* be the set of items in the optimal 2KP solution, we have

$$\sum_{j\in J^*} p_j \geqslant \frac{1}{3}\sum_{j\in S} p_j \geqslant \frac{1-\bar{\varepsilon}}{3} U_{\text{KP}} \geqslant \left(\frac{1}{3} - \varepsilon\right) \sum_{j\in S^*} p_j.$$

This shows

Proposition 2. *For any $\varepsilon > 0$, there exists a $(\frac{1}{3} - \varepsilon)$-approximation algorithm for 2KP whose running time is polynomial in the input size and in $1/\varepsilon$.*

It is very simple to adapt the above results to show that $r(U_{\text{CKP}}) = 1/4$, where U_{CKP} is the value of the continuous relaxation of (1)–(3), noting that in this case one should define a further set J_4 that contains the item which is possibly split in the continuous solution.

Moreover, consider the 2BPP, in which all items in N have to be packed into the minimum number of knapsacks (also called *bins*) of width W and height H, along with its natural one-dimensional relaxation, where items have weight $w_j h_j$ and bins have capacity WH. Letting L_{BPP} be the corresponding lower bound on the optimal 2BPP value and

$$r(L_{\mathrm{BPP}}) := \sup_{I} \left\{ \frac{OPT(I)}{L_{\mathrm{BPP}}(I)} \right\}, \tag{9}$$

Lemma 2 immediately implies

Proposition 3. $r(L_{\mathrm{BPP}}) = 3$.

Finally, note that $r(L_{\mathrm{CBPP}}) = 4$, where $L_{\mathrm{CBPP}} = \sum_{j \in N} w_j h_j / WH$ is the value of the continuous relaxation of the one-dimensional BPP defined above. This result was also shown by Martello and Vigo [20] using a different approach.

3. Enumeration schemes

The enumerative scheme for 2KP proposed by Fekete and Schepers [13] works as follows. Consider the items in the instance with the same width, height and profit values, called items of the same *type*. Let n_i be the number of items of each type i. Initially, the number of items that can be packed from each type i is in $\{0, \ldots, n_i\}$. (Actually this set can be restricted by suitable reduction rules that will not be described here.) The branching rule selects an item type i not considered in the previous levels and generates $n_i + 1$ child subproblems by imposing that, for child k ($k = 0, \ldots, n_i$), *exactly* k items of type i are packed. For each child k with $k \geqslant 1$, the method checks whether the set S of items imposed in the packing (corresponding to the constraints imposed in order to generate the child from the root) actually fits into the knapsack. For the special case of $n_i = 1$ for each type i, branching generates two children, testing feasibility only for one of them. The main novelty of the approach in [13] is the way in which the feasibility of the current set S is checked, that will be outlined in Section 3.5.

In this section, we propose alternative enumeration schemes for 2KP based on the KP relaxation (1)–(3). Namely, all schemes are defined starting from a standard branch-and-bound algorithm for KP (1)–(3), in which branching defines the current set of items S such that $\sum_{j \in S} w_j h_j \leqslant WH$ by generating child subproblems imposing that one item j not considered previously is in S and outside S (in set $\bar{S}$), respectively. Upper bounds are computed by solving the continuous relaxation of each subproblem (1)–(3) plus $x_j = 1 (j \in S)$ and $x_j = 0 (j \in \bar{S})$ (see e.g. [19] for details on branch-and-bound methods for KP). Of course in our case, the feasibility of item set S for 2KP has to be checked at some point. Moreover, this feasibility checks turn out to be by far the most time consuming part of the computation. Note that, assuming that all items have different sizes, the scheme of [13] essentially fits within this framework, in that it checks feasibility each time a new item is added to S. This policy will be referred to as "*check at all nodes*" of the branching tree. Note that each feasibility check that certifies that a set S is feasible for 2KP turns out (a posteriori) to be a waste of time if there exists some item $i \in S$ such that also $S \cup \{i\}$ is feasible. In particular, since the optimal solution values of 2KP and its KP relaxation (1)–(3) tend to be close in practice, it is natural to try to perform the feasibility test with lower frequency. The main purpose of this Section is to address this issue.

We first considered an alternative enumeration scheme in which feasibility is checked only at the *leaf nodes* of the branching tree, i.e.when no further item can be added to S, namely either $S \cup \bar{S} = N$ or

$$w_i h_i + \sum_{j \in S} w_j h_j > WH \quad \text{for each } i \in N \setminus (S \cup \bar{S}).$$

Clearly, this may have the drawback that many sets S, all supersets of a set S' already infeasible, are checked for feasibility. On the other hand, for many instances, only sets S corresponding to *maximal* KP solutions, i.e. such that

$$w_i h_i + \sum_{j \in S} w_j h_j > WH \quad \text{for each } i \in N \setminus S$$

tend to be infeasible. Hence, this policy, called "*check at leaf nodes*", may turn out to be advantageous as it may check a much smaller number of sets for feasibility with respect to the "check at all nodes" policy. We illustrate the method in detail in Section 3.2.

```
Algorithm $A_2$:
begin
  $U := U_{\mathrm{KP}}$; $optimal := FALSE$;
  repeat
    $P := \max\{\sum_{j\in R} p_j : R \subseteq N, \sum_{j\in R} w_j\, h_j \le W\, H, \sum_{j\in R} p_j \le U\}$;
    let $\mathcal{S}$ be the set of all solutions $S$ of KP (1)–(3) of value $P$;
    while $\mathcal{S} \neq \emptyset$ do
      let $S \in \mathcal{S}$;
      $\mathcal{S} := \mathcal{S} \setminus \{S\}$;
      if $U(S) \le H$ then
        $S^* := S$; $\mathcal{S} := \emptyset$; $optimal := TRUE$;
      else
        if $L(S) \le H$ then
          if $H(S) \le H$ then
            $S^* := S$; $\mathcal{S} := \emptyset$; $optimal := TRUE$;
          end if
        end if
      end if;
    end while;
    $U := P - 1$;
  until $optimal$;
end.
```

Fig. 2. Our implementation of the "check at the end of enumeration" policy.

In the "check at leaf nodes" policy we limit the feasibility checks to sets S such that $\sum_{j\in S} p_j > \sum_{j\in S^*} p_j$, where S^* is the incumbent solution. If S is feasible, we update S^* and *all* the previous feasibility checks with positive answer turn out to be a waste of time. This suggests a further enumeration scheme in which (in principle) all sets S such that $\sum_{j\in S} p_j > \sum_{j\in S^*} p_j$ are constructed and stored, where S^* is the set of items in an initial heuristic solution of 2KP, and then considered by *decreasing* values of $\sum_{j\in S} p_j$ and checked for feasibility: the first set that turns out to be feasible corresponds to the optimal 2KP solution. Since a straightforward implementation of this scheme, called "*check at the end of enumeration*", is impractical as it requires storage of too many sets, we will consider how to implement this policy in practice in Section 3.3. This method often checks for feasibility of a notably smaller number of sets with respect to the "check at leaf nodes" policy.

Finally, *hybrid* strategies that combine those presented above may be considered, one of which is presented in Section 3.4.

The problem of checking the feasibility of S is discussed in Section 3.5.

3.1. Check at all nodes

Our implementation of the "check at all nodes" policy is straightforward: we run branch-and-bound for KP (1)–(3), considering the items for branching in decreasing order of profit over weight ratio and, every time an item is added to S we compute $L(S)$ and $U(S)$ and then run two-dimensional Strip Packing Problem (2SPP) enumeration (see Section 3.5) if $L(S) \leqslant H < U(S)$, possibly updating the optimal solution S^*. The corresponding algorithm is called A_0 in the sequel.

3.2. Check at leaf nodes

In our second enumerative algorithm, called A_1, of the "check at leaf nodes" type, we compute $L(S)$ every time an item is added to S (backtracking if $L(S) > H$) and, for each leaf node associated with set S such that $L(S) \leqslant H < U(S)$ and $\sum_{j\in S} p_j > \sum_{j\in S^*} p_j$, we perform 2SPP enumeration to check the feasibility of S and possibly update the incumbent solution S^*.

```
Algorithm $A_3$:
begin
  $U := U_{\text{KP}}$; $S^* := \emptyset$;
  repeat
    define the threshold value $Z \in (\sum_{j \in S^*} p_j, U)$;
    $feasible := FALSE$;
    repeat
      let $\mathcal{S}$ be the set of the next $m$ solutions $S$ of KP (1)–(3) such that $L(S) \leq H$ and $\sum_{j \in S} p_j \in (Z, U]$;
      $\mathcal{S}' := \mathcal{S}$;
      while $\mathcal{S}' \neq \emptyset$ do
        let $S := \arg\max\{\sum_{j \in S'} p_j : S' \in \mathcal{S}'\}$;
        $\mathcal{S}' := \mathcal{S}' \setminus \{S\}$;
        if $U(S) \leq H$ then
          $S^* := S$; $\mathcal{S}' := \emptyset$; $feasible := TRUE$;
        else
          if $H(S) \leq H$ then
            $S^* := S$; $\mathcal{S}' := \emptyset$; $feasible := TRUE$;
          end if;
        end if;
      end while;
    until $\mathcal{S} = \emptyset$;
    if not $feasible$ then $U := Z$;
  until $feasible$
end.
```

Fig. 3. Our implementation of the hybrid policy.

3.3. Check at the end of enumeration

In order to implement the "check at the end of enumeration" policy in a practical way, we run branch-and-bound for KP (1)–(3) finding the corresponding optimal solution value P. Then, we re-run the branch-and-bound algorithm and check the feasibility for 2KP of the solutions of value P one after the other, in the order in which they are found. Namely, for each such solution S, we first compute $L(S)$ and $U(S)$ and then, if $L(S) \leqslant H < U(S)$, we perform 2SPP enumeration. If S is feasible, it is an optimal solution. At the end, if no solution turned out to be feasible, we decrease the current upper bound U for 2KP (initially equal to U_{KP}), and re-run the branch-and-bound for KP (1)–(3) with the additional requirement that no solution S such that $\sum_{j \in S} p_j > U$ is considered. The pseudo-code implementation of the algorithm, called A_2 in the sequel, is given in Fig. 2.

3.4. A hybrid strategy

Our hybrid enumeration scheme, called A_3 stores the first m solutions S of KP (1)–(3) such that $L(S) \leqslant H$ and $\sum_{j \in S} p_j > Z$, where Z is a *threshold value* chosen between $\sum_{j \in S^*} p_j$ (the value of the incumbent solution) and U (the value of the current upper bound). Initially, U is set to the value of the optimal solution of KP (1)–(3), whereas S^* is an initial heuristic solution (possibly, $S^* = \emptyset$). The pseudo-code implementation is given in Fig. 3. Here, "the next m solutions" stands for the solutions that are found by branch-and-bound for KP starting from the last solution considered in the previous iteration (in the first iteration, we consider "the first m solutions"). We consider the next m (initially, the first m) KP solutions of value larger than Z and check them for feasibility. If there are feasible solutions for 2KP, then the one with highest profit is optimal, otherwise Z is a valid upper bound on the optimal solution value. In the latter case, we update $U := Z$, define the new Z and iterate.

3.5. Checking the feasibility of S

The problem of checking the feasibility of S can be stated as a 2SPP, which is the variant of 2KP in which the knapsack (called *strip* in this case) has width W

Table 1
Characteristics of the instances from the literature

Name	n	W	H	$\bar{w}_j$	$\bar{h}_j$	$\bar{p}_j$	U_{KP}	OPT	$\frac{U_{\mathrm{KP}}}{OPT}$
cgcut1	16	15	10	3.25	4.50	22.75	259	244	1.06
cgcut2	23	40	70	15.13	14.17	193.43	2919	2892	1.01
cgcut3	62	40	70	3.28	5.00	26.85	2020	1860	1.09
gcut1	10	250	250	146.00	108.60	16 356.20	61 820	48 368	1.28
gcut2	20	250	250	117.50	118.35	13 728.15	62 500	59 798	1.05
gcut3	30	250	250	123.96	110.56	13 588.36	62 500	61 275	1.02
gcut4	50	250	250	117.10	122.50	14 628.16	62 500	61 380	1.02
gcut5	10	500	500	206.40	262.80	54 530.00	247 224	195 582	1.26
gcut6	20	500	500	241.90	259.80	61 602.85	249 992	236 305	1.06
gcut7	30	500	500	268.03	243.26	66 826.36	249 998	240 143	1.04
gcut8	50	500	500	228.64	245.00	56 109.24	250 000	245 758	1.02
gcut9	10	1000	1000	497.70	405.00	202 183.00	997 256	939 600	1.06
gcut10	20	1000	1000	499.15	543.40	267 768.85	999 918	937 349	1.07
gcut11	30	1000	1000	460.53	478.43	217 884.00	1 000 000	969 709	1.03
gcut12	50	1000	1000	505.18	496.74	250 439.84	1 000 000	979 521	1.02
gcut13	32	3000	3000	771.56	573.28	447 359.62	9 000 000	8 408 316[a]	1.07
wang20	42	70	40	23.21	23.83	569.28	2800	2726	1.03
okp1	50	100	100	43.40	37.52	1805.76	29 133	27 718	1.05
okp2	30	100	100	52.50	31.73	1981.80	24 800	22 502	1.10
okp3	30	100	100	57.60	30.03	2567.96	26 527	24 019	1.10
okp4	61	100	100	52.47	35.16	2394.09	33 588	32 893	1.02
okp5	97	100	100	48.47	29.55	1704.46	29 045	27 923	1.04

[a]Best solution known.

and infinite height, and all the items must be packed into the strip minimizing the overall height of the strip used. It is easy to see that also 2SPP is strongly NP-hard. We stress that our purpose here is to check the effect of different strategies to decide when to solve 2SPP instances to check feasibility of a given set S, always using a 2SPP algorithm as a *black box*.

For a given item set S, let $H(S)$ denote this minimum height. To check if S is feasible, i.e. if $H(S) \leqslant H$, we use the combinatorial lower bounds and approximation algorithms recently proposed by Fekete and Schepers [10,13] and Martello et al. [18]. In particular, given any set S of items, we will denote by $L(S)$ the best lower bound and by $U(S)$ the best heuristic solution, among those found by the procedures proposed in [10,13] and [18], for the 2SPP instance defined by the item set S. Clearly, if $U(S) \leqslant H$ set S is feasible and if $L(S) > H$ set S is infeasible. Otherwise, one must resort to enumeration to check if $H(S) \leqslant H$.

The feasibility check of the method of [13] uses lower bounds based on dual feasible functions for the one-dimensional bin packing problem [11], and performs branching based on the concept of *packing classes* [12], that represent a feasible packing of items into the knapsack without specifying the actual packing coordinates. Such a packing is described by two interval graphs, one for the horizontal and the other for the vertical direction, where vertices correspond to the items packed and edges to the overlapping of two item projections on the direction considered.

For the present work, we used the enumerative algorithm of [18] for 2SPP. This algorithm is based on a lower bound obtained by allowing each item $j \in N$ to be horizontally cut into unit-height slices, and packing the corresponding h_j slices into h_j consecutive layers on the strip. Branching is based on the notion of *envelope* and *corner points*, that specify the $O(n)$ possible positions in which the next item can be placed.

Table 2
Results on the instances from the literature

Instance		A_0			A_1			A_2			A_3		
Name	T_{FS}	T	# 2SPP	T_{2SPP}	T	# 2SPP	T_{2SPP}	T	# 2SPP	T_{2SPP}	T	# 2SPP	T_{2SPP}
cgcut1	—	0.30	43	0.30	1.47	39	1.47	1.46	39	1.46	1.46	41	1.46
cgcut2	—	1800.02	652	1799.98	1800.03	801	1799.88	533.45	576	533.01	531.93	576	531.51
cgcut3	7.47	23.76	3898	23.68	5.06	411	4.93	4.59	401	4.47	4.58	501	4.45
gcut1	—	0.00	71	0.00	0.00	3	0.00	0.01	1	0.00	0.01	30	0.00
gcut2	—	0.52	3108	0.51	0.19	272	0.14	25.75	260	0.17	0.22	1240	0.14
gcut3	—	2.80	15 854	2.67	2.16	3245	1.37	276.37	1989	0.85	3.24	11 230	1.23
gcut4	—	1800.43	372 825	1795.56	346.99	160 969	326.59	1800.00	26 918	33.04	376.52	281 760	328.10
gcut5	—	0.00	158	0.00	0.50	1	0.00	0.03	1	0.00	0.50	76	0.00
gcut6	—	0.06	1285	0.04	0.09	74	0.02	9.71	44	0.01	0.12	617	0.00
gcut7	—	1.31	8880	1.23	0.63	1260	0.23	354.50	1068	0.22	1.07	4284	0.27
gcut8	—	1202.09	805 144	1192.52	136.71	125 152	116.16	1800.01	5176	3.29	168.50	272 947	117.37
gcut9	—	0.01	196	0.01	0.09	13	0.00	0.05	4	0.00	0.08	48	0.00
gcut10	—	0.01	806	0.01	0.13	23	0.00	6.49	22	0.01	0.14	646	0.00
gcut11	—	16.72	33 429	16.39	14.76	6303	5.23	1800.01	2166	1.18	16.30	17 679	4.64
gcut12	—	63.45	183 664	61.62	16.85	29 918	13.06	1800.01	1010	0.33	25.39	72 678	13.46
gcut13	—	1800.07	20	1800.07	1800.43	1	1800.43	1800.01	0	0.00	1800.15	1	1799.20
wang20	12.51	6.75	7640	6.68	6.31	1820	6.08	17.84	832	2.06	2.72	3470	2.15
okp1	11.60	24.06	1561	24.04	25.46	1906	25.35	72.20	1683	19.87	35.84	13 996	24.53
okp2	116.24	1800.04	1694	1800.01	1800.01	798	1799.90	1535.95	1920	1141.71	1559.00	98 473	1544.89
okp3	73.03	21.36	8290	21.22	1.91	314	1.73	465.57	256	1.17	10.63	60 522	1.72
okp4	50.09	40.40	1610	40.36	2.13	150	2.08	0.85	2	0.00	4.05	14 367	1.48
okp5	40.14	1800.01	498	1800.00	1800.04	390	1800.00	513.06	701	490.75	488.27	2050	487.66

4. Experimental results

Our algorithms were implemented in ANSI C and run on a Pentium III 800 MHz. We tested our algorithms on a large set of instances from the literature. We considered instances cgcut described in [5], instances gcut described in [1] (these two classes are available in the ORLIB [3]), instance wang20 described in [22], and instances okp described in [9,13]. Instances ngcut described in [2] and hccut described in [15] are all solved within less than 1 s by our methods and the method of [13] and hence not considered in the test bed. The basic characteristics of these instances are given in Table 1, namely the instance name, the number n of items, the knapsack width W and height H, the average item width $\bar{w}_j$, height $\bar{h}_j$ and profit $\bar{p}_j$, the upper bound value U_{KP} corresponding to the optimal solution value of (1)–(3), the optimal solution value OPT, and the corresponding ratio U_{KP}/OPT, which is on average 1.07, and at most 1.28, showing that, at least for these instances, the average behavior is much better than the worst-case behavior (see Section 2).

Computational results are given in Table 2, where the results of the algorithms described above are compared with those reported by Fekete and Schepers [13] for their algorithm, which is by far the best exact algorithm in the literature (not all the instances we considered are mentioned in [13]). We set a 1800 s time limit. For each instance, we report T_{FS}, the time given in [13], T_j, the time required by each algorithm A_j ($j = 0, 1, 2, 3$), # 2SPP, the number of calls to the enumeration procedure for 2SPP, and T_{2SPP}, the overall time for these calls. Note that T_{2SPP} is *always almost equal* to the running time for A_0, A_1 and A_3, whereas for A_2 the determination of P as defined in Fig. 2 may the bottleneck when U gets notably smaller than U_{KP} (this drawback is essentially settled in A_3). Running time T_{FS} is in seconds on a Sun UltraSPARC 175 MHz (70 Mflop/s, see [7]). Since our machine has 132 Mflop/s, for a rough comparison with our algorithms the times in column T_{FS} should be halved.

Computational results show that algorithm A_3 can solve all the instances in the set with the exception of gcut13 (which is still unsolved), whereas the other algorithms exceed the time limit for 4 (A_1) or 5 (A_0 and A_2) instances. Also looking at the computing times for the instances solved by all algorithms shows that A_3 is better than the others. As to the comparison between A_3 and the algorithm in [13], the former is generally faster, but the latter appears to be more stable and is widely better on instances okp2 and okp5. In any case, the table shows that our methods are competitive with that of [13] and hence much better than the previous ones in the literature.

5. Concluding remarks

Instead of considering the area $w_j h_j$ of each item $j \in N$ as weight in the one-dimensional relaxation in Section 2, one could use the "modified" area $f(w_j)g(h_j)$, where $f(\cdot)$ and $g(\cdot)$ are *dual feasible functions* as defined in [11]. Letting $f(\cdot)$ and $g(\cdot)$ depend on the instance, it may be the case that the worst-case ratio of the corresponding bounds are much better than the ones above. We leave this as a (perhaps crucial) open problem.

Concerning the computational results, our best enumerative scheme turns out to have performances similar to those of the algorithm of [13]. The improvement with respect to the latter is conceivably due to the lower frequency of testing the feasibility of the current item set S with respect to the "check at all nodes" policy that is implicitly used in [13]. On the other hand, the algorithm of [13] is faster on some instances because the feasibility test is carried out in a more effective way. Sensible combination of the two methods is likely to lead to an overall improvement, and will be the subject of further research.

Acknowledgements

This project was partially supported by the Ministero dell'Istruzione, dell'Università e della Ricerca (MIUR) and the Consiglio Nazionale delle Ricerche (CNR), Italy. The computational experiments were executed at the Laboratory of Operations Research of the University of Bologna (LabOR). We are grateful to an anonymous referee for his/her helpful suggestions, that led to an improvement of the presentation.

References

[1] J.E. Beasley, Algorithms for unconstrained two-dimensional guillotine cutting, J. Oper. Res. Soc. 36 (1985) 297–306.

[2] J.E. Beasley, An exact two-dimensional non-guillotine cutting tree search procedure, Oper. Res. 33 (1985) 49–64.

[3] J.E. Beasley, Or-library: distributing test problems by electronic mail, J. Oper. Res. Soc. 41 (1990) 1069–1072, http://www.ms.ic.ac.uk/info.html.

[4] M.A. Boschetti, E. Hadjiconstantinou, A. Mingozzi, New upper bounds for the two-dimensional orthogonal non guillotine cutting stock problem, IMA Journal of Management Mathematics 13 (2002) 95–119.

[5] N. Christofides, C. Whitlock, An algorithm for two-dimensional cutting problems, Oper. Res. 25 (1977) 30–44.

[6] E.G. Coffman Jr., J. Csirik, G.J. Woeginger, Approximate solutions to bin packing problems, in: P.M. Pardalos, M.G.C. Resende (Eds.), Handbooks of Applied Optimization, Oxford University Press, Oxford, 2002.

[7] J. Dongarra, Performance of various computers using standard linear equations software, Technical Report cs-89-85, University of Tennessee Computer Science, 2002, available at http://www.netlib.org/utk/people/JackDongarra/papers.htm.

[8] H. Dyckhoff, G. Scheithauer, J. Terno, Cutting and Packing (C&P), in: M. Dell'Amico, F. Maffioli, S. Martello (Eds.), Annotated Bibliographies in Combinatorial Optimization, Wiley, Chichester, 1997, pp. 393–413.

[9] S.P. Fekete, J. Schepers, A new exact algorithm for general orthogonal d-dimensional knapsack problems, in: Algorithms (ESA 97), Vol. 1284, Springer Lecture Notes in Computer Science, Springer, Berlin, 1997, pp. 144–156.

[10] S.P. Fekete, J. Schepers, On more-dimensional packing II: bounds, Technical Report ZPR97-289, Mathematisches Institut, Universität zu Köln, 1997.

[11] S.P. Fekete, J. Schepers, New classes of fast lower bounds for bin packing problems, Math. Programming 91 (2001) 11–31.

[12] S.P. Fekete, J. Schepers, On more-dimensional packing I: modeling, Math. Oper. Res., to appear.

[13] S.P. Fekete, J. Schepers, On more-dimensional packing III: exact algorithms, Oper. Res., to appear.

[14] P.C. Gilmore, R.E. Gomory, Multistage cutting problems of two and more dimensions, Oper. Res. 13 (1965) 94–119.

[15] E. Hadjiconstantinou, N. Christofides, An exact algorithm for general, orthogonal, two-dimensional knapsack problems, European J. Oper. Res. 83 (1995) 39–56.

[16] E. Hadjiconstantinou, N. Christofides, An exact algorithm for the orthogonal, 2-d cutting problems using guillotine cuts, European J. Oper. Res. 83 (1995) 21–38.

[17] O.H. Ibarra, C.E. Kim, Fast approximation algorithms for the knapsack and subset sum problems, J. ACM 22 (1975) 463–468.
[18] S. Martello, M. Monaci, D. Vigo, An exact approach to the strip packing problem, INFORMS J. Comput., to appear.
[19] S. Martello, P. Toth, Knapsack Problems: Algorithms and Computer Implementations, Wiley, Chichester, 1990.
[20] S. Martello, D. Vigo, Exact solution of the two-dimensional finite bin packing problem, Manage. Sci. 44 (1998) 388–399.
[21] A. Steinberg, A strip-packing algorithm with absolute performance bound 2, SIAM J. Comput. 26 (1997) 401–409.
[22] P.Y. Wang, Two algorithms for constrained two-dimensional cutting stock problems, Oper. Res. 31 (1983) 573–586.