4OR (2008) 6:361–374
DOI 10.1007/s10288-007-0051-7

RESEARCH PAPER

A branch-and-price algorithm for the two-dimensional level strip packing problem

Andrea Bettinelli · Alberto Ceselli · Giovanni Righini

Received: 18 January 2006 / Accepted: 22 May 2007 / Published online: 19 September 2007

Abstract The two-dimensional level strip packing problem (2LSPP) consists in packing rectangular items of given size into a strip of given width divided into levels. Items packed into the same level cannot be put on top of one another and their overall width cannot exceed the width of the strip. The objective is to accommodate all the items while minimizing the overall height of the strip. The problem is $\mathcal{NP}$-hard and arises from applications in logistics and transportation. We present a set covering formulation of the 2LSPP suitable for a column generation approach, where each column corresponds to a feasible combination of items inserted into the same level. For the exact optimization of the 2LSPP we present a branch-and-price algorithm, in which the pricing problem is a penalized knapsack problem. Computational results are reported for benchmark instances with some hundreds items.

MSC classification (2000) 90C27

1 Introduction

In several industrial applications it is required to place a set of rectangular items in standard stock units. In wood and glass manufacturing, for instance, rectangular components must be cut from large pieces of material; in warehouses, the goods must be placed on shelves; in the design of newspapers layout it is needed to arrange articles and advertisements in pages of given size. These difficult combinatorial problems are often modeled as two-dimensional packing or cutting problems. Reviews on packing and cutting problems and methods can be found in Dickhoff et al. (1997) and Wäscher et al. (2006), while in Fekete and Schepers (1997, 2004a,b) the authors propose general

A. Bettinelli · A. Ceselli · G. Righini (✉)
Dipartimento di Tecnologie dell'Informazione, Università degli Studi di Milano,
Via Bramante 65, 26013 Crema, Italy
e-mail: righini@dti.unimi.it

graph-theoretical frameworks for devising bounds on multi-dimensional packing problems.

In production contexts such as clothes or paper manufacturing, a single strip of material is available and a set of items must be obtained from the strip. The aim is to cut the desired items, while minimizing the height of the strip to be used. This problem is called two-dimensional strip packing (2SPP). Recently a fully polynomial time approximation scheme for the 2SPP has been proposed (Kenyon and Rémila 2000). Martello et al. (2003) presented a branch-and-bound algorithm, which is able to solve 2SPP instances with up to 200 items at optimality in one hour of computing time.

In this paper we study a variation of the 2SPP in which a further restriction is imposed: the items must be organized into horizontal strips, indicated as *levels*; inside each level, the items cannot be put on top of one another. This problem, surveyed by Lodi et al. (2002a) and referred to as two-dimensional level strip packing problem (2LSPP), is $\mathcal{NP}$-hard in the strong sense, because it contains the bin-packing problem as a special case (Garey and Johnson 1979).

The 2LSPP can be approximated with fast heuristics, which provide also an a priori guarantee on the quality of the solution (Berkey and Wang 1987; Lodi et al. 2002a). More recently, Lodi et al. (2002b) proposed a formulation for the 2LSPP with a polynomial number of variables and constraints; the effectiveness of state-of-the-art general purpose ILP solvers makes this approach particularly appealing.

In this paper we introduce a new formulation for the 2LSPP as a set covering problem. The linear relaxation of this model is optimized with column generation, and the lower bound found in this way is used in a branch-and-price algorithm. In Sect. 2 we present our formulation and we discuss its relationship with the compact formulation of Lodi et al. (2004). In Sect. 3 we outline the main issues in the design of our branch-and-price algorithm. Finally, in Sect. 4 we report the outcome of our experimental analysis.

2 Problem formulation

In the 2LSPP we are given a strip, whose width is a positive integer W, and a set $\mathcal{N}$ of N items, where each item $j \in \mathcal{N}$ has positive integer width and height, denoted by w_j and h_j respectively. The items must be packed into levels: the sum of the widths of the items in the same level cannot exceed the width of the strip. Items in the same level cannot be piled up; hence the height of each level corresponds to the maximum height of an item in that level. We call this particular item the *leading item* and we say that the leading item *initializes* the level. Throughout the paper we assume that the items are sorted by non-decreasing height values: $h_i \leq h_j$ for each $i < j$. Hence, without loss of generality, we can state that no item i can be assigned to a level initialized by item j if $i > j$.

Lodi et al. (2002b) proposed an integer linear programming (ILP) compact formulation for the 2LSPP, which is reported here below. Each binary variable x_{ij} indicates whether item i is assigned to a level in which j is the leading item; therefore each binary variable x_{jj} indicates whether item j is a leading item. Because of the ordering of the items, we can fix each x_{ij} variable with $i > j$ to 0 and remove it from the model.

$$\min \sum_{j \in \mathcal{N}} h_j x_{jj} \tag{1}$$

$$\text{s.t.} \sum_{j \geq i} x_{ij} = 1 \qquad \forall i \in \mathcal{N} \tag{2}$$

$$\sum_{i<j} w_i x_{ij} \leq (W - w_j) x_{jj} \qquad \forall j \in \mathcal{N} \tag{3}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i \leq j \in \mathcal{N} \tag{4}$$

Constraints (2) impose that each item is assigned to a level. Constraints (3) impose that the sum of the widths of the items assigned to the same level does not exceed the width of the strip. The objective is to minimize the overall height of the strip, that is the sum of the heights of the leading items.

A set covering reformulation. A lower bound for the 2LSPP can be obtained from the model above by replacing the integrality conditions (4) with the inequalities $0 \leq x_{ij} \leq 1$. We sharpen this bound exploiting Dantzig-Wolfe decomposition (Martin 1998); for each $j \in \mathcal{N}$, let Ω_j be the set of levels complying with the capacity constraints (3) and let $\bar{\mathbf{y}}_j = (\bar{y}_1, \ldots, \bar{y}_N)$ be the generic point of Ω_j:

$$\Omega_j = \left\{ \mathbf{y} \in \mathbb{R}^N_+ | \sum_{i<j} w_i y_i \leq (W - w_j) y_j,\ y_i = 0\ \forall i > j, 0 \leq y_i \leq 1\ \forall i \in \mathcal{N} \right\}.$$

Let K_j be the index set of the integer points in Ω_j and let $\mathbf{y}^k$ be the generic integer point of Ω_j. Each point $\mathbf{y}$ in the convex hull of Ω_j can be expressed as a convex combination of the integer points of Ω_j:

$$\text{conv}(\Omega_j) = \left\{ \mathbf{y} \in \mathbb{R}^N_+ | \mathbf{y} = \sum_{k \in K_j} z_k \mathbf{y}^k,\ \sum_{k \in K_j} z_k = 1, 0 \leq z_k \leq 1\ \forall k \in K_j \right\}. \tag{5}$$

Hence we have

$$x_{ij} = \sum_{k \in K_j} y_i^k z_k \ \ \forall i, j \in \mathcal{N} \tag{6}$$

and by substitution from the linear relaxation of model (1)–(4) the following relaxation of the 2LSPP is obtained:

$$\min \sum_{j \in \mathcal{N}} \left(h_j \sum_{k \in K_j} y_j^k z_k \right)$$

$$\text{s.t.} \sum_{j \geq i} \sum_{k \in K_j} y_i^k z_k = 1 \qquad \forall i \in \mathcal{N}$$

$$\sum_{k \in K_j} z_k = 1 \qquad \forall j \in \mathcal{N} \tag{7}$$

$$0 \leq z_k \leq 1 \qquad \forall j \in \mathcal{N}, \forall k \in K_j.$$

Here, all polyhedra Ω_j have been replaced by their convex hulls. Due to this convexification, the bound found by optimizing this model dominates that of the linear relaxation of the 2LSPP.

We further elaborate on this model. First of all we remark that each K_j contains the index of a point representing an empty level: these points ($y_j^k = 0$ and therefore $y_i^k = 0 \, \forall i \leq j$) can be considered implicitly by rewriting constraints (7) as inequalities; since only points with $y_j^k = 1$ remain, the objective function can be simplified accordingly:

$$\min \sum_{j \in \mathcal{N}} \left(h_j \sum_{k \in K_j} z_k \right)$$

$$\text{s.t.} \sum_{j \geq i} \sum_{k \in K_j} y_i^k z_k = 1 \qquad \forall i \in \mathcal{N} \tag{8}$$

$$\sum_{k \in K_j} z_k \leq 1 \qquad \forall j \in \mathcal{N} \tag{9}$$

$$0 \leq z_k \leq 1 \qquad \forall j \in \mathcal{N}, \forall k \in K_j.$$

Furthermore no item is chosen more than once as a leading item in optimal solutions and hence constraints (9) are redundant and can be deleted. Finally the set partitioning constraints (8) can be replaced by set covering constraints, because it is never convenient to pack an item in more than one level.

After these manipulations the resulting model is the following:

$$\text{(MP)} \min \sum_{j \in \mathcal{N}} \left(h_j \sum_{k \in K_j} z_k \right) \tag{10}$$

$$\text{s.t.} \sum_{j \geq i} \sum_{k \in K_j} y_i^k z_k \geq 1 \qquad \forall i \in \mathcal{N} \tag{11}$$

$$0 \leq z_k \leq 1 \qquad \forall j \in \mathcal{N}, \ \forall k \in K_j. \tag{12}$$

In this linear master problem (MP) the column corresponding to each variable z_k with $k \in K_j$ represents a feasible set of items packed into a same level initialized by item j. An ILP formulation of the 2LSPP, alternative to (1)–(4), is obtained by restoring the integrality conditions $z_k \in \{0, 1\}$ in the model above.

The pricing problem. Model (10)–(12) may have a huge number of columns. Therefore a restricted master problem (RMP) involving a subset of the variables is considered and columns not included in the RMP are iteratively generated when needed.

Let λ be the vector of the non-negative dual variables associated with covering constraints (11) in a RMP optimal solution. The pricing problem we need to solve to identify new columns $(x_1, \ldots, x_N)$ is the following: $\pi(\lambda) = \min_{j \in \mathcal{N}} \{\pi_j(\lambda)\}$, where for each $j \in \mathcal{N}$

$$\pi_j(\lambda) = \min h_j y_j - \sum_{i \le j} \lambda_i y_i \tag{13}$$

$$\text{s.t.} \sum_{i<j} w_i y_i \le (W - w_j) y_j \tag{14}$$

$$y_i \in \{0, 1\} \qquad \forall i \le j. \tag{15}$$

Thus a negative reduced cost column can be generated by solving at most $|\mathcal{N}|$ binary knapsack problem instances, one for each $j \in \mathcal{N}$, obtained by setting $y_j = 1$.

Generating at each iteration as many negative reduced cost columns as possible is a common practice in branch-and-price (Desaulniers et al. 2005). However, solving a large number of knapsack problems to optimality can be unnecessary, since we just need one negative reduced cost column, provided it exists. Therefore, we solve a pricing problem in which the leading item is not fixed, but rather it must be chosen in an optimal way, that is we search for the column of minimum reduced cost for all possible choices of the leading item. The pricing problem can be rewritten in an equivalent way as follows:

$$\pi(\lambda) = \min \eta - \sum_{i \in \mathcal{N}} \lambda_i y_i \tag{16}$$

$$\text{s.t.} \sum_{i \in \mathcal{N}} w_i y_i \le W \tag{17}$$

$$h_i y_i \le \eta \qquad \forall i \in \mathcal{N} \tag{18}$$

$$y_i \in \{0, 1\} \qquad \forall i \in \mathcal{N}. \tag{19}$$

Each binary variable y_i is equal to 1 if and only if item i is assigned to the level represented by the new column. The free variable η is a penalty term. The value of η is determined by the height of the leading item of the level. The capacity constraint (17) imposes that the overall width of the level does not exceed the width of the strip.

The objective function (16) can be rewritten in maximization form:

$$\pi'(\lambda) = \max \sum_{i \in \mathcal{N}} \lambda_i y_i - \eta.$$

This pricing problem can be solved with special purpose algorithms for the penalized knapsack problem (PKP), illustrated in Ceselli and Righini (2006).

3 Branch-and-price

Branching strategy. We base our branching rule on the x variables of the compact ILP formulation (1)–(4): once an optimal MP solution z^* is obtained, a corresponding (fractional) solution x^* in terms of the original variables can be found exploiting the relation $x^*_{ij} = \sum_{k \in K_j} y^k_i z^*_k$ for each $i, j \in \mathcal{N}$, where y^k_i is the ith component of each integer vector $\mathbf{y}^k \in \Omega_j$ with $k \in K_j$.

We have adopted a two-stage branching strategy: in the first stage search-tree the branching decisions are taken on the x_{jj} variables, i.e. the leading items are chosen; in the second stage search-tree feasibility problems are solved: the non-leading items must be packed into the levels initialized by the leading items selected in the first stage, without violating width and height constraints. In both stages branching is done on the x variable whose value is closest to 0.5 and the branching variable is fixed to 0 in one branch and to 1 in the other branch; x_{jj} variables are considered in the first stage and x_{ij} variables with $i \neq j$ are considered in the second stage.

These variable fixing operations slightly change the structure of the pricing problem. In the first stage, each time a x_{jj} variable is fixed to 1, j is discarded from the set of items in the PKP optimization, and an additional KP is solved, to compute the best solution in which j is the leading item; when a x_{jj} variable is fixed to 0, it is simply discarded from the set of candidate leading items in the PKP. In the second stage the pricing problem is a KP for each level. Therefore fixing x_{ij} variables only reduces the dimension of these KP instances.

The search trees are explored in a best-bound-first order.

Initialization. In order to obtain an initial set of columns to populate the RMP, we used the well known best-fit decreasing-height (BFDH) heuristic (Lodi et al. 2002a). The items are iteratively considered from item N down to item 1 and in each iteration the current item is packed into the level with the minimum residual capacity among those that can accommodate it. If an item cannot be accommodated in this way, a new level is initialized. We implemented a simple randomized version of this heuristic (r-BFDH): a preprocessing step is added, in which r items are randomly drawn from a uniform probability distribution and the corresponding levels are initialized.

Besides running the original version of BFDH once, three r-BFDH solutions are computed for each value of r from 1 to $\lceil \sum_{i \in \mathcal{N}} w_i / W \rceil$, that is the number of levels in a fractional solution rounded up (this is a trivial lower bound on the number of levels of an optimal solution). The best solution value found in this way is also kept as an initial upper bound.

Upper bounds. We experimentally observed that the r-BFDH heuristic often provides tight bounds. Nevertheless, we incorporate a fast heuristic rounding algorithm for the set covering problem, in order to search for good integer solutions during the exploration of the search-tree. The heuristic rounding algorithm works as follows: initially, all the items are uncovered, and the columns of the RMP are sorted by non-increasing value of the corresponding z_k variables; following this order, each column k is considered: if column k represents a level containing uncovered items, the corresponding z_k variable is rounded up to 1 and each item in k is marked as covered, otherwise the z_k variable is fixed to 0.

We run this heuristic once for each node of the search tree, when the column generation process is over.

Problem reduction. Consider a generic node $\mathcal{P}$ of the first stage search-tree; let $\mathcal{N}(\mathcal{P})$ be the set of already selected leading items, $v(\mathcal{P})$ be the sum of their heights, and UB be the value of the best incumbent integer solution. For each item $j \in \mathcal{N} \setminus \mathcal{N}(\mathcal{P})$, if $v(\mathcal{P}) + h_j \geq UB$, then j can be discarded from the set of candidate leading items in node $\mathcal{P}$.

Columns deletion and re-insertion. Each time a node of the search-tree is considered, the columns in the RMP with a reduced cost higher than a given threshold are moved into a separate pool. The reduced cost of each column is computed with respect to the optimal dual solution of the ancestor node. In our implementation, the removal threshold is computed as the difference between the best known upper and lower bounds, divided by $\lceil \sum_{i \in \mathcal{N}} w_i / W \rceil$.

The columns pool is scanned at each column generation iteration: whenever a column is found to have a negative reduced cost with respect to the current dual solution, it is re-inserted into the RMP. Each column is kept into the pool for a certain number of consecutive unsuccessful checks; then it is erased. For our computational experiments we set this number to 6.

Lagrangean bounds. The bound obtained by optimizing the master problem can also be obtained by solving a Lagrangean dual problem when the set of constraints (2) is relaxed:

$$\begin{aligned} \max_{\lambda \geq 0} \ \omega(\lambda) = \min & \sum_{j \in \mathcal{N}} h_j x_{jj} - \sum_{i \in \mathcal{N}} \lambda_i \left(\sum_{j \geq i} x_{ij} - 1 \right) \\ \text{s.t.} & \sum_{i<j} w_i x_{ij} \leq (W - w_j) x_{jj} && \forall j \in \mathcal{N} \\ & x_{ij} \in \{0, 1\}. && \forall i \leq j \in \mathcal{N} \end{aligned}$$

For each set of multipliers λ this problem is analogous to the pricing problem for the set covering formulation of the 2LSPP. In fact, it decomposes into independent subproblems, one for each $j \in \mathcal{N}$:

$$\begin{aligned} \min \pi_j(\lambda) = & h_j x_{jj} - \sum_{i \leq j} \lambda_i x_{ij} \\ \text{s.t.} & \sum_{i<j} w_i x_{ij} \leq (W - w_j) x_{jj} \\ & x_{ij} \in \{0, 1\} && \forall i \leq j \in \mathcal{N}. \end{aligned}$$

Therefore, each subproblem j can be optimized by considering two cases: if the variable x_{jj} is fixed to 1, then the remaining problem is a binary knapsack; this is solved to optimality obtaining a value $\pi_j(\lambda)$. If the variable x_{jj} is fixed to 0, then each variable x_{ij} with $i < j$ must be set to 0; this yields a solution of null value. Hence, for any choice of the λ multipliers, a valid lower bound $\omega(\lambda)$ for the 2LSPP is given by

$$\omega(\lambda) = \sum_{i \in \mathcal{N}} \lambda_i + \sum_{j \in \mathcal{N}} \min\{\pi_j(\lambda), 0\}.$$

However, a key property of our pricing routine is actually to implicitly consider these π_j values to avoid the optimization of a large number of knapsack problems. In fact, the one with minimum value is computed by solving a PKP. Therefore, a lower bound

$\underline{\omega}(\lambda)$ on $\omega(\lambda)$ can be obtained by replacing each $\pi_j(\lambda)$ value with a corresponding lower bound $\underline{\pi}_j(\lambda)$.

$$\underline{\omega}(\lambda) = \sum_{i \in \mathcal{N}} \lambda_i + \sum_{j \in \mathcal{N}} \min\{\underline{\pi}_j(\lambda), 0\}.$$

We initially approximate each $\underline{\pi}_j(\lambda)$ with the value of the linear relaxation of the corresponding subproblem. These values are readily available, since they are computed in a preprocessing step by the algorithm for the PKP. Furthermore, whenever a tighter bound is computed during the optimization of the PKP, the corresponding value $\underline{\pi}_j(\lambda)$ is updated and the bound $\underline{\omega}(\lambda)$ is tightened.

Whenever, during the column generation iterations, the difference between the highest $\underline{\omega}(\lambda)$ encountered and the RMP optimal value is less than 10^{-6}, the column generation process is terminated, and the Lagrangean bound is kept as the final lower bound.

Variable fixing. We use the $\underline{\pi}_j(\lambda)$ values to fix variables. Once these values have been computed, the following reduction tests can be checked in linear time: let UB be the value of the incumbent integer solution:

- for each j such that $\underline{\pi}_j(\lambda) < 0$, if $\lceil \underline{\omega}(\lambda) - \underline{\pi}_j \rceil \geq UB$ then j can be fixed as a leading item (i.e. x_{jj} is fixed to 1);
- for each j such that $\underline{\pi}_j(\lambda) > 0$, if $\lceil \underline{\omega}(\lambda) + \underline{\pi}_j \rceil \geq UB$ then j can be discarded from the set of candidate leading items (x_{jj} is fixed to 0).

Combinatorial bound. Finally, we incorporated in our bounding procedure a combinatorial lower bound (called CUT in the remainder) proposed by Lodi et al. (2004). It consists in splitting each item in vertical strips of unit width and in filling the levels by considering these strips in order of non-increasing height. This bound dominates that given by the LP relaxation of the compact formulation (1)–(4), but no dominance relation exists with the set covering LP bound. Since we are assuming that items have been sorted in a preprocessing step, this bound can be computed in linear time.

The CUT bound is computed for each node of the search-tree before the column generation process is started. Whenever the value of an RMP optimal solution is found to be less than the value of the CUT bound, the column generation process is halted, and the CUT bound is kept as the lower bound.

4 Computational results

Our branch-and-price algorithm was implemented in C++ and compiled with a GNU C/C++ compiler version 3.2.2. We solved the restricted linear master problem with the CPLEX 8.1 implementation of the primal simplex algorithm. All our experiments were run on a Linux workstation equipped with a Pentium IV 1.6 GHz processor and 512 MB RAM. A time limit of 1 h was imposed to each test. Furthermore, the program was halted whenever the computation exceeded the amount of physical memory.

In order to assess the effectiveness of our method, we considered two data-sets for two-dimensional packing problems widely used in the literature; they are both

described in Lodi et al. (2004). The first one consists of 5 classes of instances: BENG (10 instances), CGCUT (3 instances), GCUT (4 instances), HT (9 instances) and NGCUT (12 instances). The second data-set consists of 500 instances, divided into 10 classes of 50 instances, named MV and BW. They contain instances involving up to 200 items with different types of correlation between height and width of the items.

In this section we present aggregated results. A complete report of our experiments is available in a technical report (Bettinelli and Ceselli 2007).

We considered two pricing policies: a best-negative policy, in which a PKP is solved at each column generation iteration and only the most negative reduced cost column is generated, and an all-negative policy, in which N binary knapsack problems (13)–(15) are solved and up to N negative reduced cost columns are generated. For this purpose we compared three different codes for the binary knapsack. This comparison is illustrated in Subsect. 4.1. In Subsect. 4.2 we compare different lower bounds given by different formulations for the 2LSPP. Finally, in Subsect. 4.3 we evaluate the results of our branch-and-price algorithm versus CPLEX 8.1, also comparing best-negative and all-negative pricing policies.

4.1 Tuning the all-negative pricing policy

To solve the binary knapsack problem instances, with non-integer coefficients in the objective function, we tried different techniques: (1) the MINKNAP algorithm presented in Pisinger (1997) adapted to non-integer coefficients as described in Ceselli (2003); (2) the MT1R algorithm Martello and Toth (1990), for which an implementation able to deal with non-integer coefficients is freely available on the Internet; (3) the COMBO algorithm Martello et al. (1999), which is considered the state-of-the-art code for knapsack problem instances with integer coefficients, scaling the coefficients by a factor M and rounding them down. The source code of the three algoritmhs is available on the Internet (Pisinger 2007; Martello and Toth 2007).

The third approach has two drawbacks: first, due to rounding, the value z^*_{COMBO} of the solution given by COMBO can differ by up to $N \cdot \frac{1}{M}$ from the optimal one, and therefore a valid lower bound for the pricing problem (16)–(19) can be obtained from $z^*_{\text{COMBO}} + \frac{N}{M}$. Second, setting M to very large values produces better approximations, but this may cause numerical overflow problems. Since in our data-sets there are instances with $N > 10^2$, setting M to less than 10^4 produces too weak bounds; therefore we performed experiments setting M to 10^4, 10^5 and 10^6.

In order to emphasize the effect of the knapsack algorithm implementation, we turned off the CUT bound computation, allowing more nodes of the branch-and-bound tree to be explored and more pricing iterations to be performed.

In order to catch numerical overflow problems using COMBO, we ran the branch-and-price algorithm twice for each instance: during the first run both MINKNAP and the scaled COMBO were independently used to solve each knapsack instance. In this way we could detect numerical problems when the values produced by these algorithms were different by more than $N \cdot \frac{1}{M}$. In the second run only the scaled COMBO was used and its performances were kept as the results of our experiment. By setting $M = 10^6$ we observed numerical problems in 232 of the 500 instances of

Table 1 Comparison of knapsack codes

Class	MINKNAP			MT1R			Combo $M = 10^4$				Combo $M = 10^5$			
	Inst	Gap (%)	Time(s)	Inst	Gap (%)	Time(s)	Err	Inst	Gap (%)	Time(s)	Err	Inst	Gap (%)	Time(s)
BENG	3	3.62	15.95	3	3.73	21.12		3	4.70	16.48		3	4.69	16.40
GCUT	4	0.00	7.74	4	0.00	7.92		4	0.00	9.81		4	0.00	9.30
NGCUT	12	0.00	0.04	12	0.00	0.03		12	0.00	0.04		12	0.00	0.04
CGCUT	3	0.00	1.16	3	0.00	1.24		3	0.00	0.83		3	0.00	0.82
HT	9	0.00	1.24	9	0.00	1.56		9	0.00	0.98		9	0.00	0.98
MV01	49	0.63	36.74	49	0.63	36.98	0	46	0.46	85.90	3	46	1.00	84.40
MV02	22	3.68	284.23	21	3.68	277.26	1	22	4.44	373.07	1	22	4.44	377.89
MV03	49	0.16	6.74	49	0.16	7.15	0	47	0.17	16.41	0	47	0.17	16.48
MV04	22	1.82	192.07	22	1.84	226.79	8	22	2.16	104.97	8	23	2.17	107.23
BW01	49	0.27	7.51	49	0.27	7.76	0	47	0.13	11.57	0	47	0.13	11.62
BW02	21	1.67	326.89	20	1.63	188.39	14	21	1.91	31.28	14	21	1.91	31.98
BW03	50	0.00	0.52	50	0.00	0.53	0	44	0.04	1.25	0	44	0.04	1.17
BW04	23	1.35	306.70	23	1.37	325.49	8	21	1.49	25.10	8	21	1.49	25.59
BW05	50	0.00	0.08	50	0.00	0.08	0	41	0.02	0.06	0	41	0.02	0.06
BW06	43	1.42	160.26	43	1.42	169.91	2	42	1.30	97.39	2	42	1.30	99.60

the second data-set, so we did not perform further experiments with this parameter choice.

Our computational results are reported in Table 1 for the first and the second data-set, respectively. The table is composed by four blocks, one for each branch-and-price version equipped with a particular knapsack code, as indicated in the heading row. The 'MINKNAP' and 'MT1R' blocks contain the number of instances in each class which were solved to proven optimality ('Inst'), the average optimality gap for the unsolved instances ('Gap') and the average time required to complete the computation on the solved instances ('Time'). The optimality gap is defined as $(UB - LB)/UB$, where UB and LB are, respectively the best upper and lower bounds found during the computation. In the 'COMBO' blocks we indicate also the number of instances for which numerical overflow problems were observed ('Err').

From our experiments we concluded that the scaling policy is impractical, because for both $M = 10^4$ and $M = 10^5$ numerical problems arose for more than 30 of the 500 instances of the second data-set. Furthermore considering the set of 337 instances solved to optimality using both MINKNAP and COMBO with $M = 10^4$ and in which no numerical problems were observed, the branch-and-price implementation equipped with MINKNAP took 40.75 s and the one equipped with COMBO took 58.94 s on the average. When setting $M = 10^5$ the implementation equipped with COMBO took on the average 59.49 s on the same set of 337 instances. Both MINKNAP and MT1R were effective in solving the knapsack subproblems. However, MINKNAP allowed us to solve 2 more instances to proven optimality; moreover, considering the set of 376 instances which both algorithms solved to proven optimality, the branch-and-price

algorithm with MINKNAP took on the average 74.99 s, while the one with MT1R took 84.96 s.

Therefore in our final tests with the branch-and-price algorithm using the all-negative pricing policy, we relied upon the MINKNAP adaptation for solving the binary knapsack problem instances.

4.2 Lower bounds

We compared three different lower bounds, namely the lower bound given by the linear relaxation of the set covering formulation (10)–(12), indicated hereafter with SC bound, the lower bound given by the linear relaxation of the compact formulation (1)–(4), indicated with LP bound, and the CUT lower bound. As a measure of the duality gap we considered, for each instance, the ratio $(UB - LB)/UB$, where UB is the value of the BFDH heuristic solution and LB is the value of the lower bound considered. In Table 2 we report the average values of the percentage gap for the instances in each class of the first and the second data-set, respectively. Each row of these tables corresponds to a class of instances. Since the LP and CUT values can be obtained in few hundredths of second for each instance of both data-sets, we report the average CPU time and number of iterations needed for computing the SC bound only (columns 'Time' and 'CG iter.', respectively).

Table 2 Comparison of lower bounds

Class	LP bound	CUT bound	SC bound		
	Avg. gap (%)	Avg. gap (%)	Avg. gap (%)	Time (s)	CG iter.
BENG	6.75	0.47	4.69	1.13	178.90
GCUT	14.91	10.57	0.14	0.02	3.75
NGCUT	12.57	4.21	5.10	0.01	8.17
CGCUT	4.66	4.66	6.95	0.05	20.67
HT	7.80	0.40	4.09	0.02	26.00
MV 01	8.73	6.37	2.29	0.06	3.30
MV 02	7.80	1.00	5.46	0.16	71.44
MV 03	11.95	8.97	2.96	0.07	7.40
MV 04	7.99	1.55	3.80	0.26	94.42
BW 01	11.90	9.40	2.19	0.07	5.34
BW 02	8.68	1.79	3.78	0.32	104.76
BW 03	14.57	12.17	0.69	0.10	2.52
BW 04	8.61	5.42	4.53	0.20	33.76
BW 05	19.18	17.77	0.04	0.09	2.02
BW 06	9.24	4.80	2.56	0.12	19.44

The LP bound is weaker than the CUT bound also from an experimental point of view. For the instances of the first data-set, CUT is on the average the tightest bound, while for the instances of the second data-set the SC bound is clearly superior (see, for instance, classes BW03 and BW05). On the other hand, the computation of the SC bound was often two orders of magnitude slower than that of the CUT bound.

Finally it is worth noting that the CUT and SC bounds seem to be complementary, since they are tighter for different classes of instances. This observation was one of the motivations for including the computation of both bounds in a unique lower bounding routine to solve the 2LSPP to optimality.

4.3 Solving the 2LSPP to proven optimality

We compared the performance of our branch-and-price algorithm with that of CPLEX 8.1, used as a general purpose ILP solver to optimize the compact model (1)–(4).

Table 3 contains the results for the first and the second data-set, respectively. Each entry of the table represents an average value of the instances in a class, and the class identifiers are indicated in the first column. The table is made by three blocks; each of them corresponds to the solution method indicated in the first row. In each block

Table 3 Solving the 2LSPP to proven optimality

Class	B&P, Best-neg. pricing			B&P, All-neg. pricing			CPLEX 8.1		
	Inst	Avg. gap (%)	Time(s)	Inst	Avg. gap (%)	Time(s)	Inst	Avg. gap (%)	Time(s)
BENG	10	0.00	0.02	10	0.00	0.02	3	4.96	24.47
GCUT	4	0.00	9.00	4	0.00	6.86	4	0.00	1.06
NGCUT	12	0.00	0.03	12	0.00	0.02	12	0.00	0.09
CGCUT	3	0.00	0.58	3	0.00	0.97	3	0.00	67.01
HT	9	0.00	0.04	9	0.00	0.03	9	0.00	10.14
Total	38			38			31		
MV 01	49	0.62	37.80	49	0.63	36.65	48	0.54	14.09
MV 02	46	1.01	43.59	46	1.01	91.24	25	4.26	150.01
MV 03	49	0.42	9.78	49	0.16	7.16	47	0.47	22.49
MV 04	42	0.86	161.71	42	1.04	75.66	21	3.98	173.26
BW 01	49	0.23	6.84	49	0.27	7.90	48	0.60	19.29
BW 02	37	0.92	259.43	37	0.95	349.79	21	4.35	114.12
BW 03	50	0.00	0.53	50	0.00	0.58	50	0.00	0.16
BW 04	24	1.28	254.85	23	1.28	168.95	17	2.08	94.71
BW 05	50	0.00	0.08	50	0.00	0.09	50	0.00	0.04
BW 06	43	1.51	140.10	43	1.42	129.85	34	1.20	224.74
Total	439			438			361		

we report the number of instances in each class that were solved to proven optimality (column 'Inst'), the average percentage gap between the value of the incumbent primal solution (UB) and the lower bound (LB), defined as $(UB-LB)/UB$, for the instances in which optimality was not proven (column 'Avg. gap') and the average computing time for the instances that were solved to proven optimality (column 'Time'). In the last row of the table we report the total number of instances solved to proven optimality.

The best-negative pricing policy yielded better results with respect to the all-negative policy. Using the former policy, branch-and-price solved one more instance to proven optimality. As far as computing time is concerned, a comparison was made on the 474 instances solved by both techniques: the average computing time with best-negative and all-negative was 52.43 and 61.75 s, respectively. Best-negative is especially preferable for the 'difficult' instances (those for which the overall computing time exceeds 30 min): this can be explained, because in these cases the master problem tends to grow larger and larger as time passes and hence it pays off to insert only the best columns and to keep the size of the master problem limited.

Branch-and-price solved all the instances in the first data-set, while CPLEX left a large gap on 7 of the 10 BENG instances. Moreover, branch-and-price was on the average much faster on the remaining classes. Branch-and-price performed much better than CPLEX also on the instances of the second data-set, solving more problems and consistently requiring less computing time or yielding tighter approximations, with the only exception of the 'easy' instances in the classes BW03 and BW05.

Acknowledgments This paper is based on the first author's degree thesis, that was awarded the Camerini-Carraresi prize from the Italian O.R. Society (AIRO) in 2005. The authors are grateful to Andrea Lodi for pointing out an error in a previous version of the computational results. The authors acknowledge the kind support of ACSU—Associazione Cremasca Studi Universitari to the OptLab, where this research has been done.

References

Berkey JO, Wang PY (1987) Two-dimensional finite bin-packing algorithms. J Oper Res Soc 38:423–429

Bettinelli A, Ceselli A (2007) Experimental evaluation of a branch-and-price algorithm for the bi-dimensional level strip packing problem. Technical report n. 103, Dipartimento di Tecnologie dell'Informazione, Università degli Studi di Milano

Ceselli A (2003) Two exact algorithms for the capacitated p-median problem. 4OR 1(4):319–340

Ceselli A, Righini G (2006) An optimization algorithm for a penalized knapsack problem. Oper Res Lett 34(4):394–404

Desaulniers G, Desrosiers J, Solomon MM (eds) (2005) Column Generation. Springer, Heidelberg

Dickhoff H, Scheithauer G, Terno J (1997) Cutting and packing, Wiley, New York. pp 393–413.

Fekete SP, Schepers J (1997) On higher-dimensional packing iii: Exact algorithms. Technical Report 97–290

Fekete SP, Schepers J (2004) A combinatorial characterization of higher-dimensional orthogonal packing. Math Oper Res 29:353–368

Fekete SP, Schepers J (2004) A general framework for bounds for higher-dimensional orthogonal packing problems. Math Methods Oper Res 60(2):311–329

Garey MR, Johnson DS (1979) Computers and Intractability: a guide to the theory of NP-completeness. WH Freeman, New York

Kenyon C, Rémila E (2000) A near-optimal solution to a two-dimensional cutting stock problem. Math Oper Res 25:645–656

Lodi A, Martello S, Monaci M (2002) Two-dimensional packing problems: a survey. Eur J Oper Res 141:241–252

Lodi A, Martello S, Vigo D (2002) Recent advances on two-dimensional bin packing problems. Discrete Appl Math 123:379–396

Lodi A, Martello S, Vigo D (2004) Models and bounds for two dimensional packing problems. J Comb Optim 8:363–379

Martello S, Monaci M, Vigo D (2003) An exact approach to the strip-packing problem. INFORMS J Comput 15:310–319

Martello S, Pisinger D, Toth P (1999) Dynamic programming and strong bounds for the 0–1 knapsack problem. Manage Sci 45(3):414–424

Martello S, Toth P (1990) Knapsack problems: Algorithms and computer implementations. Wiley, New York

Martello S, Toth P (2007) Online version of the book "knapsack problems": http://www.or.deis.unibo.it/knapsack.html. Last accessed 10/05/2007

Martin RK (1998) Large scale linear and integer optimization. Kluwer, Dordrecth

Pisinger D (1997) A minimal algorithm for the 0–1 knapsack problem. Oper Res 45:758–767

Pisinger D (2007) Homepage: www.diku.dk/∼pisinger/. Last accessed 10/05/2007

Wäscher G, Haussner H, Schumann H (2006) An improved typology of cutting and packing problems. Eur J Oper Res, forthcoming, 2006