Exact solution of bin-packing problems using column generation and branch-and-bound

J.M. Valério de Carvalho

Departamento Produção e Sistemas, Universidade do Minho, PT-4719 Braga, Portugal

E-mail: vc@ci.uminho.pt

We explore an arc flow formulation with side constraints for the one-dimensional bin-packing problem. The model has a set of flow conservation constraints and a set of constraints that force the appropriate number of items to be included in the packing. The model is tightened by fixing some variables at zero level, to reduce the symmetry of the solution space, and by introducing valid inequalities. The model is solved exactly using a branch-and-price procedure that combines deferred variable generation and branch-and-bound. At each iteration, the subproblem generates a set of columns, which altogether correspond to an attractive valid packing for a single bin. We describe this subproblem, and the way it is modified in the branch-and-bound phase, after the branching constraints are added to the model. We report the computational times obtained in the solution of the bin-packing problems from the OR-Library test data sets. The linear relaxation of this model provides a strong lower bound for the bin-packing problem and leads to tractable branch-and-bound trees for the instances under consideration.

Keywords: bin-packing, column generation, branch-and-bound

AMS subject classification: 90C10

1. Introduction

The bin-packing problem can be stated as follows: given a positive integer number of bins of capacity W and a list of n items of integer sizes $L = \{l_1, l_2, \ldots, l_n\}$ $(0 \le l_i \le W)$, the problem is to assign the items to the bins so that the capacity of the bins is not exceeded and the number of bins used is minimized.

The bin-packing problem belongs to the class of NP-hard problems [10], and it is not likely that there is a polynomial time algorithm to solve this problem optimally. Coffman et al. [6] did an excellent survey on this problem, particularly on approximation algorithms and their asymptotic performance ratios. Recently, Simchi-Levi [21] has shown that the first-fit decreasing and the best-bit decreasing heuristics have an absolute performance ratio of 1.5, and that this value is the best possible for the bin-packing problem, unless P = NP.

To solve the bin-packing problem to optimality, Martello and Toth [15] developed a branch-and-bound algorithm based on the following mathematical programming formulation:

$$\text{minimize } z = \sum_{i=1}^{n} y_i$$

$$\begin{aligned} \text{subject to} \quad & \sum_{j=1}^{n} l_j x_{ij} \le W y_i, && i \quad I, \\ & \sum_{i=1}^{n} x_{ij} = 1, && j \quad J, \\ & y_i = 0 \text{ or } 1, && i, \\ & x_{ij} = 0 \text{ or } 1, && i, j, \end{aligned}$$

where

$$y_i = \begin{cases} 1, & \text{if bin } i \text{ is used,} \\ 0, & \text{otherwise;} \end{cases}$$

$$x_{ij} = \begin{cases} 1, & \text{if item } j \text{ is assigned to bin } i, \\ 0, & \text{otherwise.} \end{cases}$$

A lower bound for the optimum of the bin-packing problem can be obtained from the optimum of its linear programming relaxation, which results from substituting the two last constraints for $0 \le y_i \le 1$ and $0 \le x_{ij} \le 1$.

Proposition 1.1 (Martello and Toth [15]). The lower bound provided by the linear programming relaxation of this model is equal to $\lceil \sum_{i=1}^{n} l_i / W \rceil$.

Proof. A valid solution to this model is $x_{ii} = 1$, $x_{ij} = 0$, $\forall j \ne i$, and $y_i = l_i/W$, $\forall i$. The corresponding value of the objective function is $z = \sum_{i=1}^{n} l_i/W$. As the number of bins must be integer, the lower bound will be equal to the smallest integer greater than or equal to z. □

This bound is equal to the minimum amount of space that is necessary to accommodate all the items and can be very poor for instances with large waste. That happens when all the items have a size $l_i = \lfloor W/2 + 1 \rfloor$. As W increases, the lower bound approaches 1/2.

There are many similarities between the bin-packing problem and the one-dimensional cutting stock problem. In the cutting stock problem, the items of equal size, that are usually ordered in large quantities, are grouped into orders with a required level of demand.

In the cutting stock problem, where the quantities ordered are large, the solution to the linear programming relaxation is usually used to obtain a heuristic solution of good quality to the integer problem. However, if the number of items of each size is very small, as occurs in the bin-packing problem, where the demand for a given size may be equal to one, the optimal fractional solution is useless. Rounding heuristics may lead to very poor solutions.

The one-dimensional cutting stock problem under consideration consists in determining the smallest number of rolls of width W that have to be cut in order to satisfy the demand of m clients with orders of b_d rolls of width w_d, $d = 1, 2, \ldots, m$.

A combination of orders in the width of the roll is called a cutting pattern. Let x_j be a decision variable that designates the number of rolls to be cut according to cutting pattern j. The A matrix describes the possible cutting pattern, i.e., each column $A_j = (a_{1j}, \ldots, a_{dj}, \ldots, a_{mj})^T$ defines a cutting pattern. The element a_{dj} represents the number of rolls of width w_d obtained in cutting pattern j.

The cutting stock problem is an integer programming problem that can be modelled as follows:

$$\text{minimize} \quad \sum_{j \in J} x_j \tag{1}$$

$$\text{subject to} \quad \sum_{j \in J} a_{dj} x_j \geq b_d, \quad d = 1, 2, \ldots, m, \tag{2}$$

$$x_j \geq 0, \qquad j \in J, \tag{3}$$

$$x_j \ \text{integer}, \qquad j \in J, \tag{4}$$

where J is the set of valid cutting patterns. For the cutting pattern to be valid,

$$\sum_{d=1}^{m} a_{dj} w_d \leq W, \tag{5}$$

$$a_{dj} \geq 0 \text{ and integer}, \qquad j \in J. \tag{6}$$

The number of columns in this formulation may be very large, even for moderately sized problems. It is impractical to enumerate all the columns. To tackle this problem, Gilmore and Gomory [11] introduced column generation.

For a particular case of the cutting stock problem, the binary cutting stock problem, Vance et al. [22] developed an exact procedure based on column generation and branch-and-bound. In the binary case, the demand for each order is restricted to be equal to one.

Procedures that combine column generation and branch-and-bound were used in other problems such as, for instance, a routing problem with time windows (Desrosiers et al. [5]) and the edge colouring problem (Nemhauser and Park [16]). For a discussion on column generation and branch-and-bound, see Desrosiers et al. [4].

In general, the implementation of procedures that combine the two methods has to overcome a crucial difficulty: the problem loses its "structure" as the branching constraints are added to the restricted master problem, and the type of subproblem that has to be solved at each iteration may change.

After solving the root node of the search tree, which corresponds to solving the linear programming relaxation, it may be necessary to introduce branching constraints in the restricted master problem. The column generated while solving the linear programming relaxation may be sufficient to obtain an integer solution. However, it may happen that, deep in the branch-and-bound tree, a set of columns that were not generated is necessary to obtain the optimal solution for that node, and if that set is missing, the solution obtained may be non-optimal.

The generation of columns at each node of the branch-and-bound tree occurs after the introduction of branching constraints in the restricted master problem. Under these circumstances, the subproblem is changed, and the optimal solution of the original subproblem may be a column that should not be introduced in the restricted master problem. In fact, it may happen that a particular column that was set to zero by a branching contraint turns out to be the most attractive column generated by the subproblem. To overcome this difficulty, Vance et al. [22] added generalized upper bounding constraints to the knapsack subproblem, and solved the modified problem, but eventually the subproblem completely loses its structure and has to be solved as a general integer programming problem.

Vanderbeck [23] proposed a general framework to branch in branch-and-price algorithms. Depending on the branching scheme adopted, the solution space of the subproblem is partitioned, and, for each subset, the reduced cost of the columns is evaluated taking into consideration different rewards and penalties that are based on the dual information from the branching constraints. In general, to enforce the rewards and penalties, the subproblem may have to be formulated as a general integer programming problem.

Vanderbeck also addressed the cutting stock problem. He compared several branching schemes and suggested the use of one that is based on the information that is derived from a binary representation of the cutting pattern columns. This branching rule seems to be quite robust and leads to easy amendments of the subproblem, except in situations where the number of demanded items is very small, viz., 1 or 2 units. In this case, it may happen that the subproblem can no longer be solved as a knapsack problem, making it necessary to resort to the solution of a quadratic binary knapsack problem.

In this article, we introduce an arc flow formulation with side constraints. The model has a set of flow conservation constraints and a set of constraints to ensure that the demand is satisfied. The corresponding path flow formulation is equivalent to the classical formulation for the cutting stock problem. Path formulations are usually preferred to arc formulations, because they require less computational space (see Ahuja et al. [1, chapter 17]).

We use the arc flow formulation because it allows column generation at any node in the branch-and-bound tree, and the implementation of the algorithm involves modifications to the subproblem, after solving the linear programming model, that are conceptually simpler. To accelerate the column generation procedure, instead of producing the most attractive arc, the subproblem generates sets of arcs, which correspond to valid paths, viz., cutting patterns.

Arc flow formulations usually have a large number of flow conservation constraints. A key issue is that the sets of arcs can be evaluated without explicitly considering these constraints. Actually, the model starts with all the flow conservation constraints relaxed, and only those that correspond to the arcs introduced in the restricted master problem are considered. A significant number of flow conservation constraints never have to be considered during the solution.

In section 2, we introduce the formulation and present some criteria to reduce the number of variables and also the valid inequalities that are used to tighten the formulation. In section 3, we describe the solution of the linear programming relaxation using a column generation procedure, and show the subproblem to be solved. In section 4, we present the branch-and-bound procedure, the branching rules, and the articulation of this process with the column generation procedure. In section 5, we describe some details of the implementation and the computational results obtained in the solution of some test problems.

2. Mathematical formulation

Given bins of integer capacity W and a set of different item sizes $w_1, w_2, \ldots, w_m$, the problem of determining a valid solution to a single bin can be modelled as the problem of finding a path in an acyclic directed graph with $W + 1$ vertices. Consider a graph $G = (V, A)$ with $V = \{0, 1, 2, \ldots, W\}$ and $A = \{(i, j) : 0 \leq i < j \leq W$ and $j - i = w_d$ for every $d \leq m\}$, meaning that there exists a directed arc between two vertices if there is an item of the corresponding size. The number of variables is $O(mW)$.

Consider additional arcs between $(k, k + 1)$, $k = 0, 1, \ldots, W - 1$ corresponding to unoccupied portions of the bin. There is a packing in a single bin iff there is a path between vertices 0 and W. The length of arcs that constitute the path define the item sizes to be packed.

Example 2.1. Figure 1 shows the graph associated with an instance with bins of capacity $W = 5$ and items of sizes 3 and 2. In the same figure, a path is shown that corresponds to 2 items of size 2 and 1 unit of loss.

This kind of formulation has already been used to model knapsack problems as the problem of determining the longest path in a directed graph. The approach can be traced back to Shapiro [20]. Likewise, it can be used to model bin-packing problems. If a solution to a single bin corresponds to the flow of one unit between vertices 0 and W, a path carrying a larger flow will correspond to using the same packing solution in multiple bins.

By the flow decomposition properties (see, for instance, [1]), non-negative flows can be represented by paths and cycles. The graph G is acyclic and, therefore, any

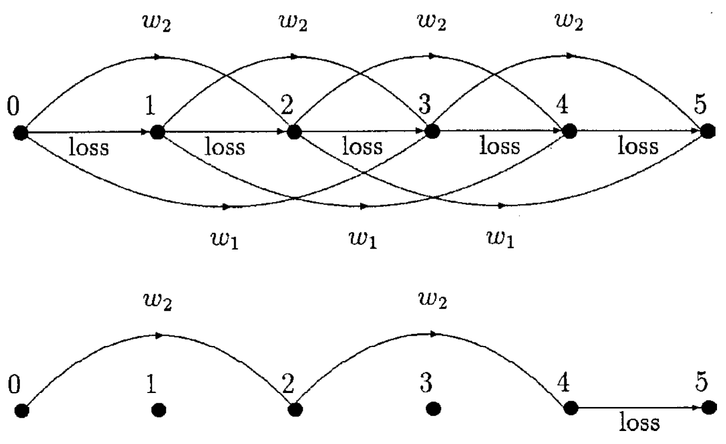

Figure 1. Graph and a cutting pattern.

flow can be decomposed into directed paths connecting the only excess node (node 0) to the only deficit node (node W).

The problem is formulated as the problem of determining the minimum flow between vertex 0 and vertex W with additional constraints enforcing that the sum of the flows in the arcs of each order must be greater than or equal to the number of items of a given size. Consider decision variables x_{ij} associated with the arcs defined above, which correspond to the number of items of size $j - i$ placed in any bin at a distance of i units from the beginning of the bin. The variable z can be seen as a feedback arc, from vertex W to vertex 0, and could also be denoted as x_{W0}. The model is as follows:

$$\text{minimize} \quad z \tag{7}$$

$$\text{subject to} \quad + \sum_{(i,j)\ \ A} x_{ij} - \sum_{(j,k)\ \ A} x_{jk} = \begin{cases} -z, & \text{if } j = 0, \\ 0, & \text{if } j = 1, 2, \quad , W-1, \\ z, & \text{if } j = W; \end{cases} \tag{8}$$

$$\sum_{(k,k+w_d)\ \ A} x_{k,k+w_d} \geq b_d, \qquad d = 1, 2, \quad , m, \tag{9}$$

$$x_{ij} \geq 0, \qquad (i,j) \quad A, \tag{10}$$

$$x_{ij} \text{ integer}, \qquad (i,j) \quad A. \tag{11}$$

The matrix that defines this problem does not have, in general, any special property that guarantees that the basic solutions obtained are integer, and it may be necessary to resort to branch-and-bound after solving the linear programming relaxation. The solution thus obtained has integer values of flow in every arc and, by the flow decomposition property referred to above, can be transformed into an integer solution to the bin packing problem.

Proposition 2.1. The linear programming flow model (7)–(10) is equivalent to the classical model (1)–(3), and hence the linear programming bounds are equal.

Proof. Apply a Dantzig–Wolfe decomposition to (7)–(10), keeping (8) and (10) in the subproblem and (7) and (9) in the master problem. Consider a polyhedron contained in the nonnegative orthant of the real numbers. According to Minkowski's theorem (see [17]), any point x of a nonempty polyhedron X can be expressed as a convex combination of the extreme points of X plus a nonnegative linear combination of the extreme rays of X:

$$X = \left\{ x \in R^n_+ : x = \sum_{q \in Q} \lambda_q x^q + \sum_{r \in R} \mu_r r^r,\ \sum_{q \in Q} \lambda_q = 1,\ \mu_r \geq 0 \right\},$$

where $Q = \{x^q\}$ is the set of extreme points of X and $R = \{r^r\}$ is the set of extreme rays of X.

The subproblem is a flow problem with a solution space that corresponds to the valid flows between node 0 to node W. The subproblem constraints define a homogeneous system, and is unbounded. Actually, if we take $z(= x_{W0})$ as a feedback arc as defined above, the solutions to the subproblem are circulation flows [1].

As referred to, any flow can be decomposed into directed paths connecting node 0 to node W. Also, the flow in a single directed path cannot be expressed as a convex combination of any two different flows. Therefore, the corresponding polyhedron has a single extreme point, the null solution, and a finite set of extreme rays, which are the directed paths, each corresponding to a valid pattern.

The subproblem will only generate extreme rays, and the substitution of patterns in (7) and (9) results in the classical model (1)–(3), which is a nonnegative linear combination of the patterns, with no convex combination constraints.

Let L_{flow} and L_{csp} be the lower bounds provided by the flow model and by the classical model, respectively. From the equivalence, it follows that $L_{flow} = L_{csp}$. □

This bound is known to be very tight. Most of the one-dimensional cutting stock instances have gaps smaller than 1, which is commonly referred to as the integer round-up property [12,13]. Nevertheless, several instances, and families of instances, are known with gaps slightly greater than 1 [9]. Using the criteria referred to below, which strengthen the formulation, the duality gap was bridged on those instances, and gaps larger than 1 have never arisen.

2.1. Enumeration of columns and reduction criteria

Using the variables presented thus far, there are many alternative solutions with exactly the same items in each bin. We may try to reduce both the symmetry of the solution space and the size of the model by considering only a subset of arcs from A.

If we search a solution in which the items are ordered in decreasing values of width, the following criteria may be used to reduce the number of arcs that are taken into account.

Criterion 1. An arc of size w_e, designated by $x_{k,k+w_e}$, can only have its tail at a node k that is the head of another arc of size w_d, $x_{k-w_d,k}$, for $w_d \geq w_e$, or, else, from node 0, i.e., the left border of the bin.

In particular, if a bin has any loss, it will appear last in the bin. A bin can never start with a loss:

Criterion 2. All the loss arcs $x_{k,k+1}$ can be set to zero for $k < w_m$.

In a bin, the number of consecutive arcs corresponding to a single item size must be less than or equal to the number of items of that size. Therefore, following criterion 1,

Criterion 3. Given any node k that is the head of another arc of size w_d ($w_d > w_e$) or $k = 0$, the only valid arcs for size w_e are those that start at nodes $k + sw_e$, $s = 0, 1, 2, \ldots, b_e - 1$ and $k + sw_e \leq W$, where b_e is the demand of items of size w_e.

Let us denote $A_{LP} \subset A$ as the set of arcs that remain after applying the above criteria.

By restricting the set of valid variables, stronger formulations may be obtained. In the classical model, each column corresponds to a valid pattern. If we search an integer solution, the maximum value of each column entry can be no larger than the corresponding demand:

$$a_{dj}^{\max} = \min \left\{ b_d, \left\lfloor \frac{W}{w_d} \right\rfloor \right\}. \tag{12}$$

The limit on the number of items can be accommodated in the knapsack solution algorithm that generates columns for the classical model. Notice that criterion 3 only approximates this limit, and it may not be possible, using the flow model, to avoid having more consecutive arcs than the value of the demand, as happens in the following example:

Example 2.2. Consider the data presented in figure 2. The path {(0, 2), (2, 3), (3, 5), (5, 7)} has three items of width 2.

Let $L_{flow'}$ be the lower bound provided by the flow model after applying the above criteria and $L_{csp'}$ the lower bounds provided by the classical model with the limits of equation (12) imposed in the cutting pattern construction for all the orders. The following relations hold:

$$L_{csp} = L_{flow} \leq L_{flow'} \leq L_{csp'}.$$

	z	x_{05}	x_{03}	x_{36}	x_{02}	x_{24}	x_{35}	x_{57}	x_{23}	x_{34}	x_{45}	x_{56}	x_{67}	
vertex 0	1	-1	-1		-1									$= 0$
1														$= 0$
2					1	-1			-1					$= 0$
3			1	-1			-1		1	-1				$= 0$
4						1				1	-1			$= 0$
5		1					1	-1			1	-1		$= 0$
6				1								1	-1	$= 0$
7	-1							1					1	$= 0$
size 1		1												≥ 1
2			1	1										≥ 3
3					1	1	1	1						≥ 2
loss									1	1	1	1	1	≥ 0

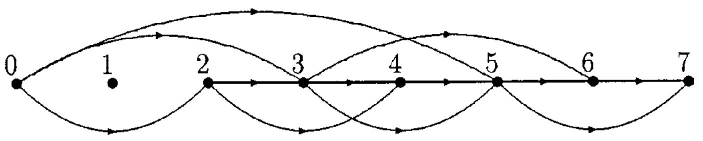

Figure 2. Linear programming model and the underlying graph.

Example 2.3. The quality of the lower bounds can be illustrated with the results obtained for the instance u250_13 of the OR-Library [2]. For this instance, the space required to accommodate all the items is equal to 101.96 bins, which gives a lower bound of 102. However, the optimum of the classical formulation for the cutting stock problem is $L_{csp} = L_{flow} = 102.0366$. It happens that $L_{flow'} = L_{csp'} = 102.0407$. The optimal integer solution is 103.

Notice that this set of criteria may not completely break the symmetry of the solution space, as happens in the following example suggested by one of the referees:

Example 2.4. Consider the data presented in figure 2. The two different paths {(0, 3), (3, 5), (5, 6), (6, 7)} and {(0, 2), (2, 3), (3, 6), (6, 7)} correspond to the same pattern.

Let z_{LP} be the optimal value of the linear programming relaxation and $\lceil z_{LP} \rceil$ the smallest integer greater than or equal to z_{LP}. After solving the linear programming relaxation, the model can be strengthened by introducing a valid inequality that forces the number of loss arcs to be greater than or equal to L_{min}.

Definition 2.1. The minimum loss $L_{min} = \lceil z_{LP} \rceil W - \sum_{d=1}^{m} w_d b_d$.

The loss will be equal to this value if the instance has the integer round-up property, as happens in the generality of the cutting stock instances.

Proposition 2.2. The constraint $\sum_{(k,k+1)\ A_{LP}} x_{k,k+1} \geq L_{min}$ is a valid inequality for the integer programming problem. According to criterion 2, this is equivalent to $\sum_{k=w_m}^{W-1} x_{k,k+1} \geq L_{min}$.

It can be shown that there exists a solution to the linear programming relaxation in which all the demand constraints are observed without slack [3]. As a corollary, this valid inequality, which imposes a lower bound on the value of the loss, forces the objective value to be integer.

In section 4, we present a procedure that makes use of the following property, which is only relevant when the instance has the integer round-up property, and which can be applied after solving the linear programming relaxation.

Proposition 2.3. If the instance has the integer round-up property and the amount of loss $L < W - w_m$, all the loss arcs $x_{k,k+1}$, $k = w_m, \ldots, W - L_{min} - 1$ can be fixed to 0.

Proof. If we search a solution with the items ordered in decreasing values of width, the loss arcs should appear at the bottom positions of the patterns, and all the loss arcs above the maximum amount of loss can be fixed to 0. This corresponds to strengthening the valid inequality presented in proposition 2.2 to $\sum_{k=W-L_{min}}^{W-1} x_{k,k+1} \geq L_{min}$. □

Both during the solution of the LP relaxation and the branch-and-bound, the subproblem can be defined as a path problem. Therefore, after solving the linear programming relaxation, it may be possible to purge all the arcs which cannot be part of a path between vertex 0 and vertex W. The following criterion can be particularly important when the amount of loss is very small.

Criterion 4. Consider the linear programming solution of an instance with the integer round-up property and the set of arcs that remain after applying proposition 2.3. Let x_{ij} be an arc in that set. If vertex j is not the origin of any valid arc (item or loss arc), the arc x_{ij} can be fixed to 0.

This criterion is applied starting from the larger index vertices back to vertex 0.

Example 2.5. Consider a bin packing instance with bins of capacity $W = 7$ and a list of items $L = \{5, 3, 3, 3, 2, 2\}$. This corresponds to a cutting stock instance with item sizes $w = (5, 3, 2)$, and quantities $b = (1, 3, 2)$. The linear programming formulation of this instance and the corresponding underlying graph are shown in figure 2. The model has a redundant constraint concerning loss that can be strengthened after solving the linear programming relaxation.

According to criterion 3, it should be pointed out that the arc x_{46} is not a valid arc for this particular instance, because the number of items of size 2 is only 2. These items must be placed after items of size 5 or 3, or at the beginning of the bin, from vertex 0. In either case, an item of size 2 can never be placed at a distance of 4 units from the left border of the bin. The number of flow conservation constraints can be smaller than the value of the capacity of the bins. In this example, there is no flow conservation constraint for vertex 1.

The solution of the linear programming relaxation has an optimal value of 2.75. This instance has the integer round-up property and its optimal solution is equal to 3. Therefore, the minimum loss $L_{min} = 3$. If we search an optimal solution with an optimal value of 3, we can fix the variables $x_{23} = x_{34} = 0$ and tighten the loss inequality, lifting its right-hand side, which becomes $x_{45} + x_{56} + x_{67} \geq 3$.

As a rule, it is not possible to establish an upper bound on the gap between the values of the optimal solutions of the integer problem and its linear programming relaxation. In the one-dimensional cutting stock problem, it happens that almost all the instances have a gap smaller than unity. Therefore, most of the instances will have a loss equal to L_{min}. However, if the instance does not have the integer round-up property, the amount of loss has to be increased by W units.

Proposition 2.4. In any integer solution, the loss $L = L_{min} + kW$, where k is a non-negative integer.

3. Linear programming relaxation

At each iteration, a subproblem is solved and a set of arcs (columns) is introduced in the restricted master problem. Let u_j, $j = 0, 1, \ldots, W$, be the dual variables associated with the flow conservation constraints and υ_d, $d = 1, 2, \ldots, m$, the dual variables associated with the demand constraints. Each column, which corresponds to arc (i, j), has only three nonzero elements, i.e., a "–1" in row i, a "+1" in row j and a "+1" in the demand row d, corresponding to the ordered width $w_d = j - i$.

Let $\pi = (u, \upsilon)$ be the dual solution of the current basis of the restricted master problem, and A_{ij} be the column corresponding to variable x_{ij}. Column A_{ij} is attractive if its reduced cost $\bar{c}_{ij} = c_{ij} - \pi A_{ij}$ is less than zero, all c_{ij} being zero. The reduced cost of a single column (arc) is equal to $\bar{c}_{ij} = -(-u_i + u_j + u_d)$.

As seen, any solution can be expressed as a nonnegative linear combination of paths. Therefore, instead of generating single arcs, sets of arcs that can be associated with a single path can be generated. Let $p = (0, a, b, \ldots, y, z, W)$ be a directed path starting at node 0 and ending at node W, and P the set of all valid paths.

To determine the most attractive path, a subproblem is solved that corresponds to the longest path in an acyclic digraph with arc costs that depend on the value of the dual variables. Let the $\bar{c}_{ij}$ be as defined above. The subproblem, which has the integrality property, takes the following form:

$$\text{minimize} \quad \sum_{(i,j)\ \ A_{LP}} \bar{c}_{ij}\, x_{ij} \tag{13}$$

$$\text{subject to} \quad + \sum_{(i,j) \in A_{LP}} x_{ij} - \sum_{(j,k) \in A_{LP}} x_{jk} = \begin{cases} -1, & \text{if } j = 0, \\ 0, & \text{if } j = 1, 2, \ldots, W-1, \\ 1, & \text{if } j = W; \end{cases} \tag{14}$$

$$x_{ij} \geq 0, \quad (i, j) \in A_{LP}. \tag{15}$$

The arcs that correspond to the variables that are already in the restricted master problem are also considered in the subproblem. There is an attractive path if the optimum is strictly less than 0. Otherwise, the solution to the restricted master problem is optimal.

At each iteration, if an arc (i, j) is part of an attractive path, the algorithm checks if the flow conservation constraints for nodes i and j, respectively, have already been considered. If not, new rows are added to the model. All the arcs in the path are also added to the restricted master problem, if not already considered. The model grows in two directions as columns and flow conservation constraints are added to it.

There is an alternative way of formulating the objective function of the subproblem, that results from the equivalence with the classical model.

Proposition 3.1. The reduced cost of a path p is equal to $\bar{c}_p = 1 - \sum_{(i,j) \in p} \upsilon_d$.

Proof. The reduced cost of a path p is equal to the sum of its arcs $\bar{c}_p = \sum_{(i,j) \in p} \bar{c}_{ij} = -\sum_{(i,j) \in p} (-u_i + u_j + \upsilon_d)$. The flow conservation dual variables cancel out for all nodes, except for the two terminal nodes. Therefore, $\bar{c}_p = -(-u_0 + u_W) - \sum_{(i,j) \in p} \upsilon_d$. The two former terms inside the parentheses correspond to a column which is symmetrical to column z and, therefore, contribute to the reduced cost with a value of –1. Each arc (i, j) contributes to the reduced cost with a value of $-\upsilon_d$. □

A set of columns is attractive if its reduced cost is less than 0. Therefore,

Corollary 3.1. The set of arcs that correspond to path p is attractive if $\sum_{(i,j) \in p} \upsilon_d > 1$.

This result has a well-known equivalent in the classical cutting stock formulation [11]. It is important because it allows the evaluation of a set of arcs (columns) without considering explicitly the flow conservation constraints. Actually, the model starts with all the flow conservation constraints relaxed, except for those corresponding to arcs in the initial starting solution.

4. Branch-and-bound

The procedure developed here is oriented to a problem where the gaps are almost always strictly smaller than one, and can be reduced to zero by introducing the valid inequality referred to in proposition 2.2. The optimization problem is solved as a sequence of decision problems in which we want to find if there is, or is not, an integer solution with an objective value equal to the smaller known lower bound, which will always be an integer value, denoted as z_{LB}.

The first decision problem to be solved is to determine if there is an integer solution of value $\lceil z_{LP} \rceil$, i.e., $z_{LB} = \lceil z_{LP} \rceil$. As is known, most of the cutting stock problem instances have the integer round-up property, meaning that almost always the solution to this decision problem will provide an optimal integer solution to the optimization problem.

However, if an integer solution is not found in the search tree, meaning that the instance does not have the integer round-up property, the lower bound has to be increased by one unit, i.e., $z_{LB} = z_{LB} + 1$, and the procedure has to be repeated. Clearly, the first integer solution found is optimal. The application of this method can only be justified in problems with minuscule gaps, and its use is not reasonable in general integer programming problems.

Branching constraints are imposed on single variables of the master problem, starting with variables that correspond to larger widths, and are placed nearer the left border of the roll, i.e., the fractional variable with the smaller value i, and to break ties the one with larger $\{j - i\}$. A depth-first search is performed, using branching constraints of the following type:

$$x_{ij} \geq \lceil x_{ij} \rceil \quad \text{and} \quad x_{ij} \leq \lfloor x_{ij} \rfloor.$$

In each node w of the search tree, we want to determine if there is any solution with objective value equal to z_{LB}. If not, the node is fathomed. This can only be done when the generation procedure fails to produce more attractive columns and the value of objective function of the restricted master problem, z^w, is strictly larger than z_{LB}.

After a branching constraint is added, the restricted master problem is reoptimized, and one of the following cases occurs:

(i) the solution is integer, with a value equal to z_{LB}, which means that it is optimal;

(ii) the solution is fractional, with a value equal to z_{LB}, making it necessary to introduce new branching constraints;

(iii) the solution has a value that is strictly greater than z_{LB}. The column generation procedure is called, trying to reach a solution with value z_{LB} leading either to case (i) or (ii). If this is not possible, the node is fathomed.

The number of decision problems to be solved is finite, as there is an absolute performance ratio for the bin-packing problem [21]. Also, the set of constraints that can be imposed in each decision problem is finite, which guarantees the finiteness of the entire solution procedure.

Figure 3 presents a flowchart for the column generation/branch-and-bound procedure.

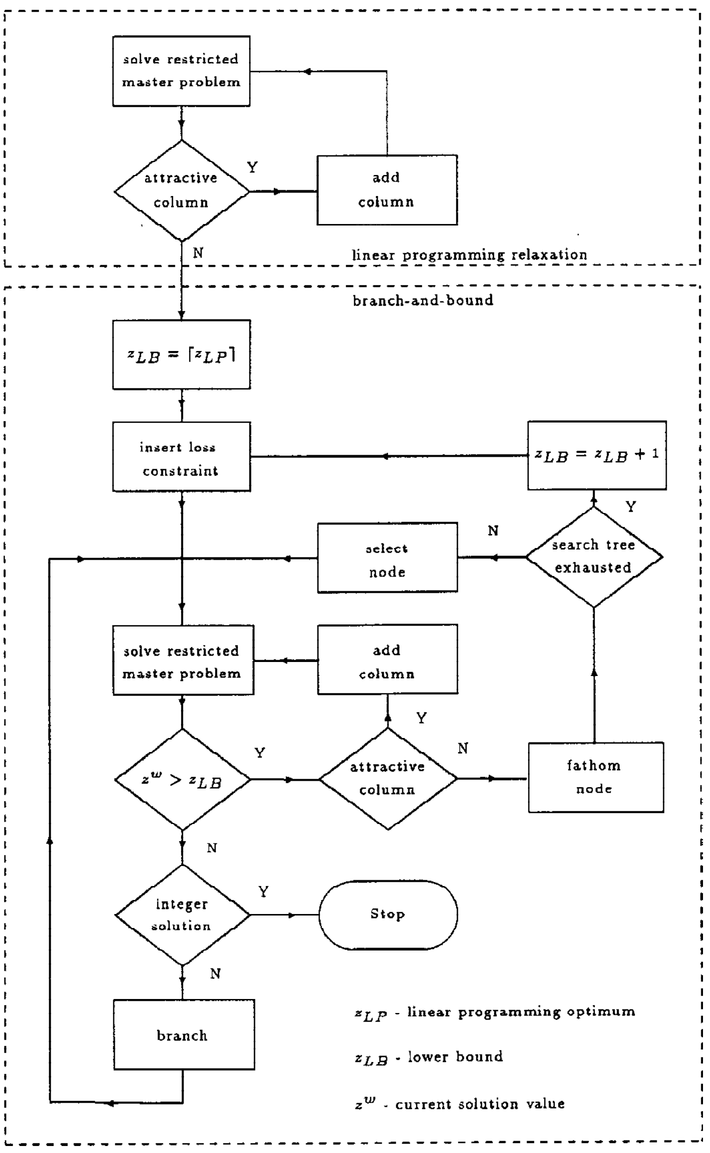

Figure 3. Column generation/branch-and-bound procedure.

4.1. Implementation

As in the framework proposed by Vanderbeck, we will use the definition of the linear programming relaxation to be solved at each node of the branch-and-bound tree to show the implementation details of the branch-and-price procedure. The dual information from the branching constraints is used to evaluate the reduced cost of the variables in the subproblem. Using the reformulation introduced in this paper that decomposes the paths into single arcs, enforcing rewards and penalties in the subproblem is straightforward. Furthermore, with the branching strategy that was used, the structure of the subproblem remains unchanged during the branch-and-bound phase. This is done at the price of having a subproblem with a pseudopolynomial number of variables, $O(mW)$.

Let $A_{BB} \subseteq A_{LP}$ be the set of arcs that remain after applying proposition 2.3 and criterion 4, which are valid for instances with the integer round-up property. Following the discussion of section 2, we did not specialize our algorithm to address instances that do not have the integer round-up property. Extensions are straightforward.

If the solution is fractional, there will be at least one fractional flow x_{ij}. The fractional solution can be eliminated by creating two branches, enforcing constraints of the following type:

$$x_{ij} \le \lfloor x_{ij} \rfloor \qquad (16)$$

and

$$x_{ij} \ge \lceil x_{ij} \rceil . \qquad (17)$$

At a branch-and-bound node w, the restricted master problem is as follows:

$$\text{minimize} \quad z \qquad (18)$$

$$\text{subject to} \quad + \sum_{(i,j)\ A_{BB}} x_{ij} - \sum_{(j,k)\ A_{BB}} x_{jk} = \begin{cases} -z, & \text{if } j = 0, \\ 0, & \text{if } j = 1, 2, \quad , W-1, \\ z, & \text{if } j = W; \end{cases} \qquad (19)$$

$$\sum_{(k,k+w_d)\ A_{BB}} x_{k,k+w_d} \ge b_d, \qquad d = 1, 2, \quad , m, \qquad (20)$$

$$x_{ij} \le \lfloor x_{ij}^l \rfloor, \qquad l \quad G^w, \qquad (21)$$

$$x_{ij} \ge \lceil x_{ij}^l \rceil, \qquad l \quad H^w, \qquad (22)$$

$$x_{ij} \ge 0, \qquad (i, j) \quad A_{BB}, \qquad (23)$$

where G^w and H^w are sets of branching constraints of type (16) and (17), respectively, for node w, and x_{ij}^l are fractional values of flow, $0 < x_{ij}^l < b_d$. Recall that the number of items for each order d, b_d, is an integer and is not restricted to be binary.

The restricted master problem is formulated in terms of arc flows. Notice that at each iteration there is a set of arcs that belong to the restricted master problem, while the remaining may be priced out singly. An arc is attractive if its reduced cost $\bar{c}_{ij} = 0 - (-u_i + u_j + \upsilon_d) < 0$. Pricing single arcs may not be very efficient. Again, we search the most attractive path.

Following the strategy described above, after adding the branching constraints and reoptimizing the restricted master problem, suppose that the optimal solution is strictly greater than z_{LB}. Let $\pi = (u, \upsilon)$, as defined, be the vector of dual variables associated with the flow conservation and demand constraints, and μ and ν the vectors of dual variables associated with the branching constraints of type (21) and (22), respectively.

The reduced cost of variable x_{ij} at node w is $\bar{c}_{ij} = 0 - (-u_i + u_j + \upsilon_d) - \sum_{l \quad G^w_{(i,j)}} \mu_l + \sum_{l \quad H^w_{(i,j)}} \nu_l)$, where $G^w_{(i,j)} \subseteq G^w$ and $H^w_{(i,j)} \subseteq H^w$ are the sets of branching constraints imposed on the specific arc (i, j). In what concerns the contributions from the dual variables of the branching constraints, the above definition is general. Actually, there may be more than one upper bound (respectively, lower bound) constraint on a given variable, but at most one of each type is observed without slack; the remaining dual variables will be null.

If there are no bounds imposed on a given variable, which certainly happens for all the variables of the restricted master problem, the summation terms of the expression of the reduced cost will be equal to 0.

The following optimality conditions do apply to the primal–dual pair: from the primal optimality conditions, for any x_{ij}, $\bar{c}_{ij} \geq 0$; from the complementary slackness conditions, if $x_{ij} > 0$, $\bar{c}_{ij} = 0$.

All basic variables have a null reduced cost. This also happens for the basic variables with a value equal to the nonzero lower or upper bound. As will be referred to in the following section, we used a software package that treats implicitly the lower and upper bounds enforced on the variables. Nevertheless, in the subproblem, all nonbasic variables at nonzero lower and upper bounds were treated as having null reduced costs. This may seem counter-intuitive in a setting where the structure of the columns does not change. It is not, if the structure of the columns that are added to the restricted master problem changes or if the variables are separate entities, as will be detailed in the comments below.

The subproblem that is used to price out attractive paths, has the integrality property, and takes the following form:

$$\text{minimize} \quad \sum_{(i,j) \quad A_{BB}} \bar{c}_{ij}\, x_{ij} \tag{24}$$

$$\text{subject to} \quad + \sum_{(i,j) \quad A_{BB}} x_{ij} - \sum_{(j,k) \quad A_{BB}} x_{jk} = \begin{cases} -1, & \text{if } j = 0, \\ 0, & \text{if } j = 1, 2, \quad , W-1 \\ 1, & \text{if } j = W; \end{cases} \tag{25}$$

$$x_{ij} \geq 0 \text{ and integer}, \qquad (i, j) \quad A_{BB}. \tag{26}$$

The arcs that correspond to the variables that are already in the restricted master problem are also considered in the subproblem. We will discuss later how the information from the branching constraints may be used to fix some more arcs to zero in the subproblem.

Since all the arcs in the restricted master problem have nonnegative reduced costs, we have the following necessary condition:

Proposition 4.1. There is an attractive path only if there is an arc with a negative reduced cost out of the restricted master problem.

Comment 1. Regeneration does not occur in the following sense: a path may contain one or more arcs that already belong to the restricted master problem (where they may be basic, nonbasic at zero level or nonbasic at a nonzero lower or upper bound), but there will always be, at least, a new variable (arc) that is brought to the main problem.

Eventually, the attractive path may contain one or more arcs with positive reduced costs, which may be nonbasic at zero level in the restricted master problem or out of the restricted master problem, if the new attractive arcs make the overall reduced cost of the path negative. If no attractive path is found, the solution to the problem at node w is optimal, as will be shown in proposition 4.2.

Comment 2. As a rule, the restricted master problem and the subproblem must be compatible. At a given node w of the branch-and-bound tree, there is a set of branching constraints that are enforced in the restricted master problem. To ensure compatibility, the variables introduced in the restricted master problem using column generation should not make feasible a point of the solution space of the master problem that violates the branching constraints imposed beforehand.

This happens naturally if we use the arc flow formulation for the restricted master problem, as the arc variables are separate entities. The existing bounds on the variables remain enforced, while the new variables enter the restricted master problem freely as nonnegative variables.

The application of the following criteria directly avoids the generation of paths that include arcs enforced to 0 in the restricted master problem.

Criterion 5. A variable (arc) that was set to zero in the restricted master problem, might be regenerated by the subproblem, as part of an attractive path. To prevent regeneration, the arc can be removed from the subproblem. This can be enforced, for instance, by setting its cost to $+\infty$.

Criterion 6. Let $A_d = \{(i, j) \quad A_{BB} : j - i = w_d\}$, $d = 1, 2, \ldots, m$, be the set of valid arcs for order d. Suppose there is a set $\hat{A}_d \subset A_d$ of arcs such that the sum of the lower bounds imposed on them sum up the required demand, that is, $\sum_{l \quad H_d^w} x_{ij} \geq \lceil \sum_{l \quad H_d^w} x_{ij}^l \rceil = b_d$, where $H_d^w \subseteq H^w$ is the set of branching constraints imposed on arcs that belong to $\hat{A}_d$. The set may be a singleton, if there is a lower bound on an arc such that $x_{ij} \geq \lceil x_{ij}^l \rceil = b_d$. There can be no optimal solution with overproduction. Therefore, all the remaining arcs $(i, j) \quad A_d \backslash \hat{A}_d$ can be removed from the subproblem.

Comment 3. Formulations based on arc flows are usually denoted as “original formulations”, while the formulations that use the deferred column generation of paths are denoted as “path formulations”. In this setting, the arc flow formulation can be seen as the original formulation of the classical model of Gilmore–Gomory for the cutting-stock problem. It is interesting to analyse what would be the structure of the columns to be introduced in the restricted master problem at a given node w of a branch-and-bound node of the path flow formulation, and how that structure follows from the compatibility issue addressed in comment 2.

It is accepted that it is a good strategy to impose the branching constraints on the variables of the original formulation [4]. Suppose that the following strategy is used to solve the bin-packing problem: a path formulation is used to solve the restricted master problem, the branching constraint are imposed on the flow variables of the original formulation, and the subproblem is solved using the arc flow formulation.

An attractive valid path found by the subproblem is a sequence of arcs that define an appropriate integer vector with the number of items for each demand. Suppose that the attractive path, besides the new attractive arcs, had one or more arc variables on which upper and/or lower bounds had been imposed beforehand in the restricted master problem. If there is a bound on the flow of a given arc variable, it follows from comment 2 that the path variables in the restricted master problem that contain that arc should altogether obey the bound imposed beforehand.

Therefore, the path column to be introduced in the restricted master problem should not enter freely as a nonnegative variable, but it should obey all the branching constraints enforced on the arcs that already belonged to the restricted master problem. Besides the elements pertaining to the demand, the path column should have a +1 in the appropriate branching constraint. Clearly, the reduced cost of such path columns depends on the values of the dual variables of the branching constraints.

The converse of proposition 4.1 is not true. The master problem is formulated in terms of arc variables. To prove the validity of the above approach, which is based on the generation of arcs from attractive paths, it is necessary to prove the following result:

Proposition 4.2. If $z^w > z_{LB}$ and there are no more attractive paths, the generation process may be abandoned, and the corresponding branch-and-bound node w fathomed, even if there are still attractive arcs, with a negative reduced cost, out of the restricted master problem.

Proof. The result follows from the representation of the solution of the master problem as a nonnegative linear combination of paths. The value of a single arc that is out of the restricted master problem will not increase from the zero level, if the remaining arcs of the path make the path unattractive, or even if the arc is not part of any path, due to the removal of arcs during the branch-and-bound phase. □

It is well known that there are optimal degenerate extreme points that have bases that are optimal and bases that still have attractive variables. Intuitively, if we ignore the attractive arcs, we may spare some degenerate pivots before concluding that the extreme point is optimal.

5. Computational results

The performance of the algorithm was tested using the instances from the OR-Library Benchmark Database, kept by John Beasley at the Imperial College, UK [2], which are available by e-mail. The instances were a contribution of E. Falkenauer, who used them to test a genetic algorithm [8]. The instances are divided into two main classes: the *u* class and the *t* class, which stand for uniform and triplet class, respectively.

The class designated by *u* has instances with a bin capacity of $W = 150$ and items with integer sizes, drawn from a uniform distribution between 20 and 100. There are four groups of 20 instances, each with 120, 250, 500 and 1000 items, respectively.

The class denoted by *t* has instances with a bin capacity of $W = 1000$, with items of integer sizes between 250 and 500. These instances are designated by triplets because the optimal solution has exactly three items per bin, which fulfill the capacity of the bin. They were designed to be especially hard to solve. The optimal solution to each instance is known in advance because the size of the items is generated as follows: the size of the first item is drawn uniformly between 380 and 490, which gives a space S left of between 510 and 620; the second item is drawn from a uniform distribution between 250 and $S/2$; and, finally, the size of the third item is equal to the remaining space. This procedure is repeated as many times as there are bins in the optimal solution [8]. There are four groups of 20 instances, each with 60, 120, 249 and 501 items, respectively.

The procedure developed is based on the XMP routines, described in Marsten [14], which use the revised simplex method and the factorization of the inverse. Lower and upper bounds are dealt with implicitly, which enables an easy implementation of the branch-and-bound procedure. Experiments were run in a 120 MHz Pentium, with 16 Mbytes of RAM memory. The algorithms were coded in FORTRAN and compiled using Salford FTN77 under DBOS [19].

During a pre-processing phase, all the possible arcs are enumerated according to the criteria presented in section 2. Then, the initial solution is set. The columns selected are those that correspond to a first-fit decreasing heuristic (FFD).

The subproblem is solved by a dynamic programming reaching procedure [7]. Let A^w be the set of valid arcs in node w of the branch-and-bound tree. Let $F(x)$ be the minimum marginal cost for a path from node 0 to node x. We have the following set of recurrence relations: $F(0) = 0$; $F(x) = \min_{(i,j)\ A^w}\{F(x - w_d) + \bar{c}_{ij}\}$, for all $x = 1, 2, \ldots, W$. A single path is generated in each call to the subproblem.

In table 1, we present the data related to the solution of all the bin-packing instances. The values shown are averages over the 20 instances in each class. The meaning of each column is as follows:

m	:	number of different item sizes;
m_{init}	:	starting number of constraints after the heuristic;

Table 1

Average solution times.

	m	m_{init}	m_{max}	arc_{init}	sp_{lp}	arc_{lp}	sp_{bb}	arc_{bb}	$node_{bb}$	t_{pp}	t_{lp}	t_{bb}	t_{tot}
u120	63.2	121.5	183.4	91.7	69.3	137.3	5.4	7.7	17.9	0.05	2.96	1.21	4.22
u250	77.3	160.4	200.8	147.1	60.8	113.3	13.7	16.1	31.9	0.05	3.27	2.66	5.98
u500	80.8	187.5	208.9	205.9	29.3	51.8	24.1	35.3	49.5	0.07	2.20	4.66	6.93
u1000	81.0	199.0	211.4	247.6	14.5	25.6	29.7	44.1	64.7	0.11	1.57	5.77	7.45
t60	50.0	110.3	181.6	59.9	97.8	168.9	5.0	5.3	2.0	0.06	3.50	0.58	4.14
t120	86.1	199.5	301.8	118.6	163.8	286.2	69.5	71.1	20.7	0.08	12.54	14.33	26.95
t249	140.1	355.9	483.4	233.8	242.0	403.7	197.4	199.5	58.3	0.09	37.33	85.42	122.85
t501	194.2	557.6	676.5	410.6	344.4	489.8	299.7	301.0	119.0	0.16	85.59	274.94	360.69

m_{max}	: number of constraints upon termination;
arc_{init}	: number of arc variables after the heuristic;
sp_{lp}	: number of calls to subproblem while solving the linear programming relaxation;
arc_{lp}	: number of columns generated while solving the linear programming relaxation;
sp_{bb}	: number of calls to subproblem during the branch-and-bound phase;
arc_{bb}	: number of columns generated during the branch-and-bound phase;
$node_{bb}$	: number of nodes explored in the branch-and-bound phase;
t_{pp}	: preprocessing time (seconds);
t_{lp}	: linear relaxation solution time (seconds);
t_{bb}	: branch-and-bound time (seconds);
t_{tot}	: total time (seconds);
z_{lp}	: LP lower bound;
z^*	: optimum value.

The solution times for each instance are shown in tables 2–9. As a general rule for the instances in the u classes, it happens that the smaller the gap $(z^* - z_{lp})$, the larger the work of the branch-and-bound phase.

The evaluation of the performance of this algorithm and of the size of the instances it can solve should be done very carefully. As a matter of fact, large economies of scale may result from grouping several items of the list with the same size into a single order of the cutting stock problem. In terms of this algorithm, a better measure to describe the size of the instance is the number of different item sizes. It is this value that defines the size of the linear programming problems to be solved. However, it seems fair to compare the performance of the column generation/branch-and-

Table 2

Uniform, 120 items.

	m	m_{init}	m_{max}	arc_{init}	sp_{lp}	arc_{lp}	sp_{bb}	arc_{bb}	$node_{bb}$	t_{pp}	t_{lp}	t_{bb}	t_{tot}	z_{lp}	z^*
u120_00	58	115	185	92	66	121	0	0	15	0.05	2.69	0.61	3.35	47.266	48.00
u120_01	59	120	192	91	80	147	0	0	5	0.05	3.52	0.16	3.74	48.049	49.00
u120_02	61	126	186	95	64	138	2	3	22	0.05	3.08	1.04	4.18	45.293	46.00
u120_03	68	124	181	91	82	158	0	0	19	0.05	3.30	0.71	4.07	48.626	49.00
u120_04	62	122	195	89	57	124	0	0	17	0.05	2.58	0.71	3.35	49.085	50.00
u120_05	61	119	176	89	68	141	0	0	22	0.05	3.19	1.04	4.29	47.490	48.00
u120_06	65	125	180	96	74	154	1	1	21	0.05	3.30	0.88	4.23	47.580	48.00
u120_07	64	129	177	103	63	123	0	0	13	0.05	2.42	0.55	3.02	48.660	49.00
u120_08	67	120	174	89	70	145	12	19	14	0.05	2.96	2.64	5.66	49.912	50.00
u120_09	64	127	177	100	66	138	4	5	12	0.05	2.80	0.88	3.74	45.800	46.00
u120_10	64	116	190	85	91	170	0	0	20	0.05	3.90	0.99	4.95	51.280	52.00
u120_11	60	118	176	89	66	118	0	0	21	0.05	2.36	0.61	3.02	48.393	49.00
u120_12	63	124	176	92	61	140	63	93	29	0.00	3.19	6.54	9.73	47.867	48.00
u120_13	62	119	195	89	45	110	0	0	23	0.00	2.20	0.88	3.07	48.013	49.00
u120_14	61	121	192	92	77	138	0	0	22	0.05	3.13	0.93	4.12	49.170	50.00
u120_15	63	121	187	92	70	143	0	0	18	0.05	2.97	0.66	3.68	47.384	48.00
u120_16	64	119	187	86	67	122	7	8	17	0.05	2.47	0.88	3.41	51.333	52.00
u120_17	64	116	181	83	63	126	0	0	12	0.05	2.47	0.38	2.91	51.500	52.00
u120_18	66	127	186	97	89	157	0	0	21	0.05	3.84	0.82	4.72	8.382	49.00
u120_19	68	122	175	93	67	133	20	25	16	0.05	2.80	2.31	5.16	48.864	49.00
	63.2	121.5	183.4	91.7	69.3	137.3	5.4	7.7	17.9	0.05	2.96	1.21	4.22		

Table 3

Uniform, 250 items.

	m	m_{init}	m_{max}	arc_{init}	sp_{lp}	arc_{lp}	sp_{bb}	arc_{bb}	$node_{bb}$	t_{pp}	t_{lp}	t_{bb}	t_{tot}	z_{lp}	z^*
u250_00	71	152	187	148	48	96	0	0	24	0.05	2.47	0.93	3.46	98.553	99.00
u250_01	78	161	211	150	60	110	0	0	34	0.05	3.73	1.70	5.49	99.027	100.00
u250_02	76	156	202	137	77	139	0	0	34	0.05	3.79	1.54	5.38	101.422	102.00
u250_03	79	169	205	158	37	84	6	13	31	0.05	2.42	1.87	4.34	99.427	100.00
u250_04	76	163	200	147	56	115	9	13	45	0.05	3.18	3.19	6.43	100.613	101.00
u250_05	78	162	199	146	71	135	5	7	29	0.05	3.79	2.58	6.43	100.827	101.00
u250_06	78	164	211	149	39	96	0	0	39	0.05	2.91	2.09	5.05	101.027	102.00
u250_07	77	157	195	148	67	122	19	26	18	0.05	3.35	3.24	6.65	102.885	103.00
u250_08	78	158	194	141	62	121	98	109	31	0.05	3.02	7.53	10.60	104.918	105.00
u250_09	80	169	208	153	75	125	0	0	39	0.05	4.29	1.81	6.15	100.201	101.00
u250_10	77	156	203	139	64	119	1	1	24	0.05	3.46	1.15	4.67	104.395	105.00
u250_11	80	165	203	152	84	136	2	2	27	0.05	3.95	2.04	6.04	100.713	101.00
u250_12	76	154	180	135	88	132	69	72	32	0.05	3.30	7.14	10.50	104.977	105.00
u250_13	75	155	208	142	58	115	5	7	41	0.05	3.41	2.58	6.04	102.041	103.00
u250_14	79	160	209	152	59	105	0	0	27	0.05	3.35	1.32	4.73	99.167	100.00
u250_15	78	158	197	139	65	117	55	61	34	0.05	3.13	5.94	9.12	104.861	105.00
u250_16	76	157	197	148	52	102	0	0	26	0.11	2.64	1.10	3.85	96.513	97.00
u250_17	76	162	206	151	44	90	0	0	42	0.05	2.80	1.76	4.62	99.167	100.00
u250_18	76	163	193	148	63	109	3	5	25	0.05	3.46	1.92	5.44	99.700	100.00
u250_19	81	166	208	159	46	99	2	6	36	0.05	2.86	1.76	4.67	101.360	102.00
	77.3	160.4	200.8	147.1	60.8	113.3	13.7	16.1	31.9	0.05	3.27	2.66	5.98		

Table 4

Uniform, 500 items.

	m	m_{init}	m_{max}	arc_{init}	sp_{lp}	arc_{lp}	sp_{bb}	arc_{bb}	$node_{bb}$	t_{pp}	t_{lp}	t_{bb}	t_{tot}	z_{lp}	z^*
u500_01	81	191	209	208	20	34	101	170	62	0.05	1.38	13.52	14.95	200.847	201.00
u500_02	80	187	209	208	27	57	6	14	43	0.11	2.09	2.53	4.72	201.440	202.00
u500_03	81	190	204	208	38	62	35	52	54	0.05	2.58	7.75	10.38	203.813	204.00
u500_04	80	189	212	201	35	67	0	0	59	0.05	2.58	2.47	5.11	205.113	206.00
u500_05	81	174	214	184	39	71	0	0	51	0.05	2.64	2.20	4.89	205.087	206.00
u500_06	81	181	208	188	35	67	130	172	61	0.05	2.09	14.62	16.76	206.906	207.00
u500_07	80	186	204	191	35	63	139	183	56	0.11	2.31	15.71	18.13	203.980	204.00
u500_08	81	186	206	204	31	51	25	46	62	0.05	1.92	5.71	7.69	195.680	196.00
u500_09	81	193	214	215	31	54	0	0	66	0.11	2.64	3.07	5.82	201.060	202.00
u500_10	81	184	214	198	27	46	1	3	63	0.05	2.03	2.75	4.84	199.067	200.00
u500_11	81	188	208	202	25	49	10	15	46	0.05	1.87	3.68	5.61	199.427	200.00
u500_12	81	192	205	222	28	46	2	2	33	0.05	2.14	1.65	3.85	198.620	199.00
u500_13	81	187	202	218	13	24	7	15	32	0.05	1.54	2.04	3.63	195.587	196.00
u500_14	81	185	214	211	38	57	0	0	59	0.05	3.19	3.08	6.32	203.027	204.00
u500_15	80	195	213	213	32	50	0	0	64	0.11	2.47	2.58	5.16	200.133	201.00
u500_16	81	185	214	209	31	59	0	0	52	0.05	2.47	2.25	4.78	201.007	202.00
u500_17	81	190	208	217	29	47	2	2	26	0.05	2.47	1.32	3.84	197.427	198.00
u500_18	81	183	209	199	22	44	0	0	29	0.05	1.87	1.10	3.02	201.293	202.00
u500_19	81	193	206	209	27	50	4	5	32	0.11	1.87	1.59	3.57	195.633	196.00
	80.8	187.5	208.9	205.9	29.3	51.8	24.1	35.3	49.5	0.07	2.20	4.66	6.93		

Table 5

Uniform, 100 items.

	m	m_{init}	m_{max}	arc_{init}	sp_{lp}	arc_{lp}	sp_{bb}	arc_{bb}	$node_{bb}$	t_{pp}	t_{lp}	t_{bb}	t_{tot}	z_{lp}	z^*
u1000_00	81	203	211	252	9	15	6	12	55	0.11	1.26	3.41	4.78	398.427	399.00
u1000_01	81	198	214	249	16	26	1	2	45	0.11	1.92	2.25	4.29	405.253	406.00
u1000_02	81	198	214	239	17	31	7	13	71	0.11	1.76	4.67	6.54	410.200	411.00
u1000_03	81	197	211	238	22	36	109	153	84	0.05	1.87	13.19	15.11	410.867	411.00
u1000_04	81	199	206	262	12	18	7	10	37	0.11	1.43	2.09	3.63	396.740	397.00
u1000_05	81	198	209	239	18	30	7	17	31	0.11	1.65	2.36	4.12	398.493	399.00
u1000_06	81	203	214	273	7	15	0	0	69	0.11	1.26	2.91	4.29	394.207	395.00
u1000_07	81	201	214	268	10	19	0	0	67	0.11	1.54	3.19	4.84	403.160	404.00
u1000_08	81	195	211	241	21	35	4	7	45	0.11	1.81	2.80	4.72	398.433	399.00
u1000_09	81	200	211	252	14	29	121	185	87	0.11	1.54	15.22	16.87	396.927	397.00
u1000_10	81	197	214	236	13	22	5	9	64	0.11	1.32	2.97	4.39	399.340	400.00
u1000_11	81	198	207	253	14	18	4	13	53	0.11	1.48	2.31	3.90	400.520	401.00
u1000_12	81	199	214	253	14	26	1	4	73	0.17	1.65	3.52	5.33	392.240	393.00
u1000_13	81	206	214	262	12	17	1	2	68	0.11	1.38	3.19	4.67	395.273	396.00
u1000_14	81	206	211	262	13	24	197	256	114	0.11	1.38	21.81	23.30	393.887	394.00
u1000_15	81	201	208	249	14	25	13	23	39	0.11	1.76	3.85	5.71	401.807	402.00
u1000_16	81	194	214	240	12	24	0	0	63	0.11	1.92	2.91	4.95	403.027	404.00
u1000_17	81	193	208	230	17	36	99	150	99	0.11	1.26	15.22	16.59	403.800	404.00
u1000_18	81	192	212	214	17	37	4	8	65	0.11	1.48	3.07	4.67	398.193	399.00
u1000_19	81	201	212	240	17	29	7	18	64	0.11	1.76	4.39	6.26	399.333	400.00
	81.0	199.0	211.4	247.6	14.5	25.6	29.7	44.1	64.7	0.11	1.57	5.77	7.45		

Table 6

Triplets, 60 items.

	m	m_{init}	m_{max}	arc_{init}	sp_{lp}	arc_{lp}	sp_{bb}	arc_{bb}	$node_{bb}$	t_{pp}	t_{lp}	t_{bb}	t_{tot}	z_{lp}	z^*
t60_00	50	109	179	59	104	170	0	0	1	0.05	3.57	0.05	3.68	20.000	20.00
t60_01	56	117	196	60	107	191	27	29	18	0.05	4.45	5.00	9.50	20.000	20.00
t60_02	48	110	183	60	113	190	0	0	0	0.05	4.12	0.00	4.18	20.000	20.00
t60_03	49	110	181	60	89	164	10	10	4	0.05	2.97	0.71	3.74	20.000	20.00
t60_04	48	108	178	60	88	157	0	0	0	0.11	2.86	0.00	2.97	20.000	20.00
t60_05	50	110	175	59	86	152	0	0	0	0.05	2.86	0.00	2.91	20.000	20.00
t60_06	51	109	182	60	91	160	16	16	7	0.05	3.08	1.87	5.00	20.000	20.00
t60_07	50	111	181	60	106	175	11	12	3	0.05	3.79	1.10	4.95	20.000	20.00
t60_08	49	108	171	60	87	153	0	0	0	0.05	3.19	0.00	3.24	20.000	20.00
t60_09	49	109	193	60	126	204	0	0	0	0.11	4.61	0.00	4.73	20.000	20.00
t60_10	53	114	194	60	110	189	10	11	2	0.05	4.18	0.99	5.22	20.000	20.00
t60_11	46	108	179	60	94	163	0	0	0	0.05	3.18	0.00	3.24	20.000	20.00
t60_12	53	113	184	60	96	164	12	13	2	0.05	3.52	0.71	4.29	20.000	20.00
t60_13	51	112	183	60	94	166	3	3	1	0.05	3.63	0.33	4.01	20.000	20.00
t60_14	50	109	175	60	91	156	0	0	0	0.05	3.08	0.05	3.19	20.000	20.00
t60_15	51	113	181	60	98	169	11	12	2	0.05	3.68	0.77	4.51	20.000	20.00
t60_16	46	108	184	60	93	169	0	0	0	0.05	3.02	0.00	3.08	20.000	20.00
t60_17	48	110	172	60	83	148	0	0	0	0.06	2.75	0.00	2.80	20.000	20.00
t60_18	50	109	165	60	89	146	0	0	0	0.05	3.02	0.00	3.08	20.000	20.00
t60_19	51	110	195	60	112	193	0	0	0	0.06	4.39	0.00	4.45	20.000	20.00
	50.0	110.3	181.6	59.9	97.8	168.9	5.0	5.3	2.0	0.06	3.50	0.58	4.14		

Table 7

Triplets, 120 items.

	m	m_{init}	m_{max}	arc_{init}	sp_{lp}	arc_{lp}	sp_{bb}	arc_{bb}	$node_{bb}$	t_{pp}	t_{lp}	t_{bb}	t_{tot}	z_{lp}	z^*
t120_00	86	193	285	115	153	271	84	84	24	0.05	10.55	18.02	28.63	40.000	40.00
t120_01	85	198	292	118	159	275	79	81	21	0.05	12.36	17.80	30.22	40.000	40.00
t120_02	91	197	298	117	165	289	85	86	24	0.05	13.13	14.39	27.58	40.000	40.00
t120_03	86	199	302	118	169	297	36	37	15	0.05	12.80	10.38	23.24	40.000	40.00
t120_04	92	209	319	120	171	304	112	112	31	0.05	13.46	23.02	36.54	40.000	40.00
t120_05	88	200	302	119	156	275	81	84	23	0.05	11.32	15.88	27.25	40.000	40.00
t120_06	86	201	306	120	175	300	90	93	29	0.05	14.23	16.87	31.16	40.000	40.00
t120_07	87	198	300	119	176	294	57	58	19	0.11	13.52	12.70	26.32	40.000	40.00
t120_08	86	202	307	120	166	294	59	60	20	0.05	12.91	14.12	27.09	40.000	40.00
t120_09	85	197	295	119	147	262	84	86	19	0.05	11.10	14.72	25.88	40.000	40.00
t120_10	83	197	294	120	147	262	67	68	18	0.11	10.55	12.36	23.02	40.000	40.00
t120_11	86	204	311	119	190	323	60	64	17	0.05	15.49	13.74	29.29	40.000	40.00
t120_12	82	191	281	117	150	259	18	19	5	0.11	10.60	4.73	15.44	40.000	40.00
t120_13	87	199	303	118	158	279	57	60	23	0.11	12.14	14.89	27.14	40.000	40.00
t120_14	83	197	299	118	163	284	79	81	27	0.11	12.14	17.09	29.34	40.000	40.00
t120_15	81	195	287	119	139	248	32	33	10	0.11	9.40	5.88	15.39	40.000	40.00
t120_16	87	204	310	120	166	287	87	89	21	0.05	12.42	14.23	26.70	40.000	40.00
t120_17	90	208	319	120	174	308	60	61	15	0.11	15.05	12.25	27.42	40.000	40.00
t120_18	85	201	311	117	174	306	68	69	21	0.11	13.35	11.70	25.16	40.000	40.00
t120_19	87	201	316	120	179	307	96	96	32	0.11	14.34	21.81	36.27	40.000	40.00
	86.1	199.5	301.8	118.6	163.8	286.2	69.5	71.1	20.7	0.08	12.54	14.33	26.95		

Table 8

Triplets, 249 items.

	m	m_{init}	m_{max}	arc_{init}	sp_{lp}	arc_{lp}	sp_{bb}	arc_{bb}	$node_{bb}$	t_{pp}	t_{lp}	t_{bb}	t_{tot}	z_{lp}	z^*
t249_00	134	332	451	217	233	406	165	168	52	0.11	38.73	75.50	114.34	83.00	83.00
t249_01	140	361	476	236	214	359	167	169	51	0.11	31.16	59.56	90.82	83.000	83.00
t249_02	139	356	475	240	235	377	223	225	69	0.06	30.71	93.35	124.12	83.000	83.00
t249_03	142	361	486	236	233	390	243	245	69	0.11	37.64	81.92	119.67	83.000	83.00
t249_04	134	343	474	224	251	412	118	120	40	0.11	35.50	47.69	83.30	83.000	83.00
t249_05	145	368	504	242	249	419	170	174	61	0.05	39.84	94.23	134.12	83.000	83.00
t249_06	138	348	485	232	226	393	217	218	58	0.05	30.88	71.48	102.42	83.000	83.00
t249_07	137	349	481	229	253	427	224	227	58	0.11	40.88	106.59	147.58	83.000	83.00
t249_08	139	355	486	236	254	410	141	144	49	0.11	41.32	65.61	107.04	83.000	83.00
t249_09	141	360	502	239	235	403	216	217	62	0.11	37.80	98.52	136.43	83.000	83.00
t249_10	140	356	485	234	253	413	191	192	59	0.11	39.45	75.11	114.67	83.000	83.00
t249_11	141	360	487	230	249	414	236	238	57	0.11	37.47	77.25	114.83	83.000	83.00
t249_12	141	354	467	236	219	367	146	149	58	0.11	33.46	84.07	117.64	83.000	83.00
t249_13	141	354	479	231	240	398	296	296	68	0.11	37.36	107.20	144.67	83.000	83.00
t249_14	145	363	491	238	242	405	180	181	52	0.05	37.03	84.12	121.21	83.000	83.00
t249_15	142	360	490	234	255	431	250	254	70	0.06	44.23	105.00	149.29	83.000	83.00
t249_16	144	367	496	238	255	428	158	162	52	0.11	39.73	88.84	128.68	83.000	83.00
t249_17	145	359	484	235	255	422	282	283	73	0.11	42.03	139.51	181.65	83.000	83.00
t249_18	138	353	472	237	237	384	131	133	51	0.11	33.63	84.01	117.75	83.000	83.00
t249_19	136	359	497	232	252	416	194	196	57	0.05	37.75	68.90	106.70	83.000	83.00
	140.1	355.9	483.4	233.8	242.0	403.7	197.4	199.5	58.3	0.09	37.33	85.42	122.85		

Table 9

Triplets, 501 items.

	m	m_{init}	m_{max}	arc_{init}	sp_{lp}	arc_{lp}	sp_{bb}	arc_{bb}	$node_{bb}$	t_{pp}	t_{lp}	t_{bb}	t_{tot}	z_{lp}	z^*
t501_00	190	533	603	391	261	331	294	294	131	0.16	60.44	178.90	239.50	167.000	167.00
t501_00	190	533	653	301	312	453	286	287	117	0.16	71.04	247.25	318.46	167.000	167.00
t501_01	192	545	695	393	369	550	126	127	76	0.17	92.96	187.20	280.33	167.000	167.00
t501_02	190	552	668	409	343	482	334	335	127	0.16	80.11	311.65	391.92	167.000	167.00
t501_03	199	566	681	416	314	458	392	394	134	0.11	80.50	326.43	407.03	167.000	167.00
t501_04	195	568	695	424	334	502	338	340	126	0.16	91.65	354.45	446.27	167.000	167.00
t501_05	195	560	686	410	335	493	350	353	128	0.16	87.09	243.30	330.55	167.000	167.00
t501_06	196	560	669	416	352	496	258	261	126	0.16	91.98	267.09	359.23	167.000	167.00
t501_07	192	559	671	419	353	475	207	207	104	0.16	83.13	206.81	290.11	167.000	167.00
t501_08	196	568	705	411	402	555	299	300	126	0.17	94.83	337.70	432.70	167.000	167.00
t501_09	189	561	675	418	357	505	301	302	129	0.11	88.85	254.55	343.52	167.000	167.00
t501_10	190	548	665	409	338	479	343	344	111	0.16	78.46	263.46	342.09	167.000	167.00
t501_11	195	560	694	416	381	541	320	321	126	0.16	94.73	315.05	409.95	167.000	167.00
t501_12	189	553	663	407	326	451	162	162	94	0.16	73.13	179.18	252.48	167.000	167.00
t501_13	198	576	692	427	332	493	365	365	126	0.16	90.66	305.00	395.83	167.000	167.00
t501_14	203	564	687	413	317	472	311	311	122	0.22	71.15	280.00	351.37	167.000	167.00
t501_15	197	558	676	411	350	503	378	379	124	0.17	99.83	348.80	448.80	167.000	167.00
t501_16	197	547	622	397	259	333	256	257	109	0.11	60.22	182.14	242.47	167.000	167.00
t501_17	196	573	696	421	410	547	214	218	112	0.17	91.37	221.92	313.46	167.000	167.00
t501_18	193	560	690	411	363	526	401	403	131	0.17	101.76	358.52	460.44	167.000	167.00
t501_19	192	542	648	393	341	482	353	354	131	0.11	88.35	308.24	396.70	167.000	167.00
	194.2	557.6	676.5	410.6	344.4	489.8	299.7	301.0	119.0	0.16	85.59	274.94	360.69		

bound algorithm on instances with a given number of different item sizes with the performance of other exact algorithms on instances with the same given number of items.

Comparisons were made with MTP, developed by Martello and Toth [15], a branch-and-bound algorithm based on the formulation described above. This algorithm can be preset to stop after a pre-determined number of backtracks, producing the best incumbent heuristic solution found so far. Experiences run on the same machine show that MTP fails to produce the optimal solution after 1500000 backtracks in 14 out of 20 instances in the t60 class and in all instances of the t120 class. The average time used to perform that number of backtracks is about 370 seconds for the t60 class (60 items) and over 750 seconds for the t120 class (120 items). The incumbent solutions were, on average, at a distance of 10% from the optimum.

As shown, the column generation/branch-and-bound algorithm consistently solves the instances of the t120 class which have, on average, 86.1 different item sizes, and those of the t249 class which have, on average, 140.1 different item sizes. The algorithm also performs better on the uniform class. In this case, the advantage clearly results from the grouping of items of the same size into the same order. MTP fails to produce the optimal solution in 3 out of 20 problems in the u120 class, in 11 out of 20 problems in the u250 class, and in all other larger instances.

Vanderbeck [24] also gave results from a computational study of bin-packing and cutting stock problems. Vanderbeck makes a comparison between the results obtained using his solution approach and the results we presented in a previous version of this paper. We must mention that the times that we reported were obtained in a machine that is about 5 times slower than the one used this time.

Even though the machines are different, judging from the counters, Vanderbeck found the results for the two approaches to be comparable for the instances in the *u* class. For the triplets, Vanderbeck encountered some difficulties due to changes in the structure of the subproblem, which happened in the situations that were referred to in section 1. Nevertheless, he was able to solve all the instances of the quadratic knapsack subproblem by amending the procedure for the solution of a linear knapsack problem. For these instances, Vanderbeck also found the computational results to be comparable [24].

6. Conclusions and further work

We presented a new model and an exact procedure for the one-dimensional cutting stock problem, and its application to the solution of bin-packing problems. The algorithm combines column generation and branch-and-bound, and the dimensions of the restricted master problem grow in two directions. As the algorithm proceeds, sets of variables are added to the main problem and the flow conservation constraints corresponding to those variables, that were relaxed at the beginning, are added to the model, if not already present.

In this formulation, large economies of scale may result from grouping several items with the same size into the same order, when bin-packing problems with integer data are considered. The measures that better describe the dimensions of the instances that can be solved are the number of different item sizes and the value of the capacity of the bins. The sizes of the subproblem and of the restricted master problem depend crucially on these values.

As the linear programming relaxation of the formulation used is very tight, the branch-and-bound trees are tractable for the instances under consideration. Larger instances were solved, with not only a larger number of items, but also with a larger number of different item sizes.

Nevertheless, it may be possible to obtain better computational results. The arc formulation used to solve the master problem has a linear relaxation that is not as strong as the one provided by the path formulation. Furthermore, the size of the basis increases as the arc variables are added to the master problem. A solution approach as the one sketched in comment 3, with an adequate branching rule that partitions the solution space more evenly, will be presented in a subsequent paper.

Acknowledgements

We are indebted to Professor J. Desrosiers for suggesting proposition 2.1, and providing the line for its proof, for sharpening the discussion of the LP bounds, as well as for many constructive comments, that led to a clearer presentation. We are also grateful to François Vanderbeck for his valuable comments. This work was partially supported by the Algoritmi Research Center of the University of Minho, and was developed in the Industrial and Systems Engineering Group.

References

[1] R. Ahuja, T. Magnanti and J. Orlin, *Network Flows: Theory, Algorithms and Applications*, Prentice-Hall, Englewood Cliffs, NJ, 1993.

[2] J.E. Beasley, OR-library: Distributing test problems by electronic mail, Journal Operational Research Society 41(1990)1069–1072.

[3] J.M. Valério de Carvalho, A model for the one-dimensional cutting-stock problem, Working Paper, 1996.

[4] J. Desrosiers, Y. Dumas, M. M. Salomon and F. Soumis, Time constrained routing and scheduling, in: *Handbooks in Operations Research & Management Science 8: Network Routing,* Elsevier Science, 1995.

[5] J. Desrosiers, F. Soumis and M. Desrochers, Routing with time windows by column generation, Networks 14(1984)545–565.

[6] E.G. Coffman. Jr., M.R. Garey and D.S. Johnson, Approximation algorithms for bin-packing – an updated survey, in: *Algorithm Design for Computer System Design,* Springer, Berlin, 1984.

[7] E.V. Denardo, *Dynamic Programming Models and Applications,* Prentice-Hall, NJ, 1982.

[8] E. Falkenauer, A hybrid grouping genetic algorithm for bin packing, International Journal of Computers and Operations Research (1995), to appear.

[9] M. Fieldhouse, The duality gap in trim problems, SICUP Bulletin 5 (November 1990).

[10] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H.Freeman, San Francisco, 1979.

[11] P.C. Gilmore and R.E. Gomory, A linear programming approach to the cutting stock problem – part ii, Operations Research 11(1963)863–888.

[12] O. Marcotte, The cutting stock problem and integer rounding, Mathematical Programming 33(1985) 82–92.

[13] O. Marcotte, An instance of the cutting stock problem for which the rounding property does not hold, Operations Research Letters 4(1986)239–243.

[14] R.M. Marsten, The design of the XMP linear programming library, ACM Transactions on Mathematical Software 7(1981)481–497.

[15] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*, Wiley, New York, 1990.

[16] G. Nemhauser and S. Park, A polyhedral approach to edge coloring, Operations Research Letters 10(1991)315–322.

[17] G. Nemhauser and L. Wolsey, *Integer and Combinatorial Optimization*, Wiley, New York, 1988.

[18] C.H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Englewood Cliffs, NJ, 1982.

[19] Salford, *Salford Software*, 1995.

[20] J.F. Shapiro, Dynamic programming algorithms for the integer programming problem I: The integer programming problem viewed as a knapsack type problem, Operations Research 16(1968)103–121.

[21] D. Simchi-Levi, New worst-case results for the bin-packing problem, Naval Research Logistics 41(1994)579–585.

[22] P. Vance, C. Barnhart, E. L. Johnson and G. L. Nemhauser, Solving binary cutting stock problems by column generation and branch-and-bound, Computational Optimization and Applications 3 (1994)111–130.

[23] F. Vanderbeck, On integer programming decomposition and ways to enforce integrality in the master, Research Papers in Management Studies, 1994–1995, No. 29 (revised May 1996), University of Cambridge, 1996.

[24] F. Vanderbeck, Computational study of a column generation algorithm for binpacking and cutting stock problems, Research Papers in Management Studies, 1996, No. 14, University of Cambridge, 1996.