Models and Algorithms for Three-Stage Two-Dimensional Bin Packing [1]

Jakob Puchinger [*] and Günther R. Raidl

Institute of Computer Graphics and Algorithms
Vienna University of Technology
Favoritenstraße 9–11/1861, 1040 Vienna, Austria
Phone: +431 58801 18611
Fax: +431 58801 18699
{puchinger|raidl}@ads.tuwien.ac.at

Preprint submitted to Elsevier Science *30 November 2005*

Abstract

We consider the three-stage two-dimensional bin packing problem (2BP) which occurs in real-world applications such as glass, paper, or steel cutting. We present new integer linear programming formulations: models for a restricted version and the original version of the problem are developed. Both only involve polynomial numbers of variables and constraints and effectively avoid symmetries. Those models are solved using CPLEX. Furthermore, a branch-and-price (B&P) algorithm is presented for a set covering formulation of the unrestricted problem, which corresponds to a Dantzig-Wolfe decomposition of the polynomially-sized model. We consider column generation stabilization in the B&P algorithm using dual-optimal inequalities. Fast column generation is performed by applying a hierarchy of four methods: (a) a fast greedy heuristic, (b) an evolutionary algorithm, (c) solving a restricted form of the pricing problem using CPLEX, and finally (d) solving the complete pricing problem using CPLEX. Computational experiments on standard benchmark instances document the benefits of the new approaches: The restricted version of the integer linear programming model can be used to quickly obtain near-optimal solutions. The unrestricted version is computationally more expensive. Column generation provides a strong lower bound for 3-stage 2BP. The combination of all four pricing algorithms and column generation stabilization in the proposed B&P framework yields the best results in terms of the average objective value, the average run-time, and the number of instances solved to proven optimality.

1 Introduction

The two-dimensional bin packing (2BP) problem occurs in different variants in important real-world applications such as glass, paper, and steel cutting. A set of two-dimensional, differently sized items is given. They have to be packed into (or cut out of) bins (or sheets of some raw-material) of a fixed size. The aim is to minimize the number of needed bins (sheets) and, therefore, the waste. A recent survey on 2D packing problems is given in Lodi et al. [1], an annotated bibliography on cutting and packing is presented by Dyckhoff et al. [2].

In practice, there are often special requirements on the cutting/packing patterns. Here, we consider in particular orthogonal *guillotine* cuts; i.e., pieces are always rectangular and may only be cut horizontally or vertically from one border to the opposite one. Furthermore, real-world cutting machines are often constructed such that the sheets are processed in a certain number of *stages*.

* corresponding author

[1] This work is supported by the RTN ADONET under grant 504438 and the Austrian Science Fund (FWF) under grant P16263-N04.

In each stage either horizontal or vertical cuts can be performed, but never both. Pieces having passed a stage may not be put back to a previous stage. These conditions limit the nesting of horizontal and vertical cuts and, thus, the maximum height of the cutting tree of each bin. In this article we focus on three-stage problems, where the first stage is only able to perform horizontal cuts, the second only vertical cuts, and the third again only horizontal cuts; see Fig. 1 for an example of a feasible cutting pattern. The three-stage restriction is, for example, typical for glass manufacturing [3,4]. The packing problem considered here is a two-dimensional rectangular Single Bin Size Bin Packing Problem (SBSBPP) as defined in the typology of Wäscher et al. [5].

The next section gives a short overview on previous work related to the considered problem. The combination of exact and heuristic methods for solving difficult problems is a central concern of the work presented here. Section 3 defines the three-stage two-dimensional bin packing (3-stage 2BP) problem in a more formal way. We then develop an integer linear programming (ILP) model for a restricted version of 3-stage 2BP and extend it to a model for the unrestricted case in Sec. 4. These models involve only $O(n^2)$ and $O(n^3)$ variables, the number of constraints is bounded by $O(n)$ and $O(n^3)$ respectively, and symmetries are effectively avoided. To our knowledge, this is the first polynomial-sized ILP for 3-stage 2BP.

In Sec. 5, we describe an alternative approach based on a Dantzig-Wolfe decomposition [6]: the 3-stage 2BP problem is reformulated as a set covering problem and fast column generation is performed. We also introduce dual subset inequalities in order to derive dual constraints to stabilize the column generation process. Section 6 describes how branching is performed in order to get optimal integer solutions. Furthermore, the column generation process is described in detail: Columns are generated by using a hierarchy of four methods, namely a greedy heuristic, an evolutionary algorithm, and a restricted as well as an unrestricted ILP for the pricing problem. In Sec. 7, computational experiments are described and analyzed. We finally summarize our work and draw conclusions in Sec. 8.

2 Previous Work

Most of the simple algorithms for 2BP are of greedy nature. Items are placed one by one and never reconsidered again. One- and two-phase algorithms are presented in the literature [1]. In one-phase algorithms, the items are directly placed into the bins, whereas the two-phase algorithms first partition the items into levels (stripes) which are then assigned to bins by solving a one-dimensional bin packing problem. Berkey and Wang [7] described the classical Finite First Fit heuristic, which is a greedy one-phase algorithm. Items are sorted by decreasing heights. The first item initializes the first level in the first bin and defines the level's height. Each following item is added to the first level to which it fits respecting the bin's width. If there is no such level, a new level is initialized in the first bin into which it fits. And, if there is no such bin, a new bin is initialized with the new level. Within a level, items are never stacked.

The approach of formulating a packing or cutting problem as a set covering problem, which we will pursue in Sec. 5, originates in the work of Gilmore and Gomory [8]. They propose this technique for the one dimensional cutting stock problem. This formulation introduces a variable for each possible cutting pattern of a single bin. Since, in general, the number of these variables increases exponentially with respect to the problem size, not all variables are explicitly considered and column generation is performed. In [9], Gilmore and Gomory applied the same basic technique to two-dimensional stock cutting. The major difference lies in the method for solving the pricing problem, i.e., in the way of determining promising patterns/variables that may improve a current solution. With respect to stage-constraints, only two-stage guillotineable patterns are considered. Unfortunately, this approach cannot directly be extended to three or more stages in an efficient way.

A faster variant of the Gilmore and Gomory approach is proposed by Oliveira and Ferreira [10], where also three-stage and general multi-stage cutting stock problems are considered. For solving the pricing problem, a greedy heuristic is first applied in the hope that it quickly finds a suitable variable. Only if this heuristic fails, a slower exact algorithm is used.

Monaci and Toth [11] present a set covering based heuristic approach for bin-packing problems. In a first phase, columns (i.e. patterns) are generated using greedy and fast constructive heuristics, in a second phase the associated set-covering instance is solved by means of a Lagrangian-based heuristic algorithm.

More recently the 2-stage 2BP was considered in Lodi et al. [12]. They introduced the first compact ILP model involving only a polynomial number of variables and constraints. This work is the basis for the ILPs we introduce in Sec. 4 for 3-stage 2BP.

Particular real-world three-stage cutting problems with specific additional properties were treated in Vanderbeck [13] and Puchinger et al. [4]. Vanderbeck solves a three-stage two-dimensional cutting stock problem, where the main objective is to minimize waste, but other issues such as aging stock pieces, urgent or optimal orders, and fixed setup costs are also considered. His solution approach uses nested decomposition of the problem and a recursive use of column generation. Puchinger et al. consider a 3-stage 2BP problem appearing in glass cutting, where specific additional constraints with respect to the order of items are imposed. The problem is heuristically solved using an evolutionary algorithm (EA) based on an order representation, specific recombination and mutation operators, and a greedy decoding heuristic. In one variant, branch-and-bound is occasionally applied to locally optimize parts of a solution during decoding. A more general overview of evolutionary algorithms for cutting and packing problems is given by Hopper [14].

The following two algorithms exactly solve the more general 2BP problem where also non-guillotineable patterns of rectangular items are allowed. Martello and Vigo [15] describe a two-level branch-and-bound algorithm. Items are assigned to bins by an outer decision tree. Possible packing patterns for the bins are generated by trying a heuristic first; if it fails to place all necessary items, it tries to find a pattern by an inner enumeration scheme. A hybrid branch-and-price / constraint programming algorithm has been proposed by Pisinger and Sigurd [16,17]. They use the column generation principle of Gilmore and Gomory and solve the specific pricing problem by means of constraint programming.

Lodi et al. [18] describe a general heuristic framework applicable to several variants of 2BP. Tabu search is used to assign the items to bins, and cutting patterns for individual bins are obtained by different inner heuristics.

Preliminary results with particular respect to column generation by means of an evolutionary algorithm have been published by the authors in [19].

3 Three-Stage Two-Dimensional Bin Packing

In the *2BP* problem, we are given a set of n rectangular items having individual heights h_i and widths w_i, $i = 1, ..., n$. The objective is to pack them into a minimum number of rectangular bins, each having height H and width W. Items may not overlap and we do not consider rotation. We assume $0 < w_i \leq W,\ 0 < h_i \leq H$.

A feasible solution for *3-stage 2BP* consists of a set of *bins*, where each bin is partitioned into a set of *stripes*, each stripe consists of a set of *stacks*, and each stack consists of a set of *items* having equal width. The packing patterns of such a solution can always be transformed into the so-called *normal form* by moving each item to its uppermost and leftmost position, so that void space may only appear at the bottom of stacks, to the right of the last stack in each stripe and below the last stripe; see Fig. 1 for an example. In the sequel we consider only patterns in normal form.

We assume the items are sorted so that $h_1 \geq h_2 \geq \ldots \geq h_n$. The order of the items within each stack does not affect the feasibility and the objective value of a solution, so the items can always be sorted according to their indices. A solution may contain a maximum of n stacks. We label each stack with the index of the highest item it contains, i.e., with its smallest item index. Similarly, a solution has at most n stripes, and a stripe's label corresponds to the label of its highest stack. Finally, a maximum of n bins is needed, and

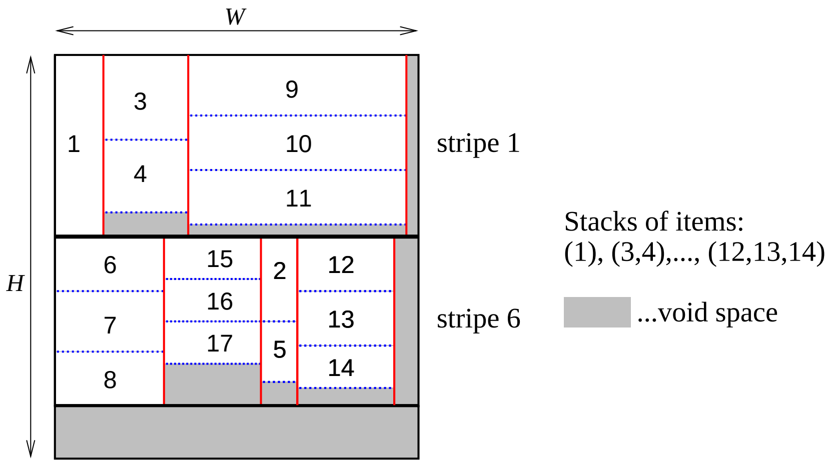

Fig. 1. A three-stage cutting pattern for one bin in normal form.

we label each of the bins with the smallest index of the stripes it contains. In the example shown in Fig. 1, the stack labels are 1, 3, 9, 6, 15, 2, and 12, the stripes are labeled 1 and 6, and the bin's label is 1.

4 Integer Linear Programming Models

In this section, new ILP models for different versions of 3-stage 2BP are presented. These formulations are based on concepts for 2-stage 2BP from Lodi et al. [12].

4.1 Restricted Three-Stage Two-Dimensional Bin Packing

We first describe a model for *restricted 3-stage 2BP* where the highest stack of each stripe, which determines the label of the stripe, always consists of a single item. Therefore, the highest item (i.e., the one with the smallest index) of a stripe defines its height. This restriction helps to avoid some difficulties when calculating the total height of all stripes contained in a bin. An overview of other primal restriction strategies can be found in [20].

The model uses the following variables.

- $\alpha_{j,i} \in \{0,1\}$, $j = 1,\dots,n,\ \ i = j,\dots,n$:
  set to one if and only if (iff) item i is contained in stack j
- $\beta_{k,j} \in \{0,1\}$, $k = 1,\dots,n,\ \ j = k,\dots,n$:
  set to one iff stack j is contained in stripe k
- $\gamma_{l,k} \in \{0,1\}$, $l = 1,\dots,n,\ \ k = l,\dots,n$:
  set to one iff stripe k is contained in bin l

The restricted 3-stage 2BP problem can now be stated as the following ILP.

$$\text{minimize} \quad \sum_{l=1}^{n} \gamma_{l,l} \tag{1}$$

$$\text{subject to} \quad \sum_{j=1}^{i} \alpha_{j,i} = 1, \ \forall i = 1, \ldots, n \tag{2}$$

$$\alpha_{j,i} = 0, \ \forall j = 1, \ldots, n-1, \forall i > j \mid w_i \neq w_j \vee h_i + h_j > H \tag{3}$$

$$\sum_{k=1}^{j} \beta_{k,j} = \alpha_{j,j}, \quad \forall j = 1, \ldots, n \tag{4}$$

$$\sum_{i=j}^{n} h_i \alpha_{j,i} \ \leq \sum_{k=1}^{j} h_k \beta_{k,j}, \quad \forall j = 1, \ldots, n-1 \tag{5}$$

$$\sum_{j=k}^{n} w_j \beta_{k,j} \ \leq \ W \beta_{k,k}, \quad \forall k = 1, \ldots, n-1 \tag{6}$$

$$\sum_{l=1}^{k} \gamma_{l,k} = \beta_{k,k}, \quad \forall k = 1, \ldots, n \tag{7}$$

$$\sum_{k=l}^{n} h_k \gamma_{l,k} \leq \ H \gamma_{l,l}, \quad \forall l = 1, \ldots, n-1 \tag{8}$$

$$\alpha_{j,i} \in \{0,1\}, \ j = 1, \ldots, n, \ i = j, \ldots, n \tag{9}$$

$$\beta_{k,j} \in \{0,1\}, \ k = 1, \ldots, n, \ j = k, \ldots, n \tag{10}$$

$$\gamma_{l,k} \in \{0,1\}, \ l = 1, \ldots, n, \ k = l, \ldots, n \tag{11}$$

The objective function (1) minimizes the number of used bins. Note that bin l is used iff $\gamma_{l,l} = 1$. Equations (2) ensure that each item i has to be packed once. The fact that the items packed into the same stack must have identical width and that the total height of any pair of stacked items must not exceed H is guaranteed by (3). In an actual implementation it is only necessary to consider the variables $\alpha_{j,i}$ for which $w_i = w_j$ and $h_i + h_j \leq H$. Here, however, we keep all variables in our model for the sake of clarity. Each *used* stack j—i.e., stack j contains item j and thus $\alpha_{j,j} = 1$—is packed exactly once into a stripe k according to equations (4). Constraints (5) ensure that the height of each stack j—i.e., the total height of all items contained in stack j—never exceeds the height of the associated stripe k, which is identical to item k's height due to our special problem restriction. The constraints (4) and (5) together further imply that no items may be packed into an unused stack. Constraints (6) guarantee that the bins' width W is not exceeded by any stripe k and that no stacks are packed into unused stripes ($\beta_{k,k} = 0$). Equations (7) force each used stripe k to be packed into exactly one bin. Finally, constraints (8) guarantee that each bins' height H is not exceeded by the total height of the packed stripes and that no stripes are packed into unused bins ($\gamma_{l,l} = 0$).

In total, the ILP uses $O(n^2)$ variables and $O(n)$ constraints, considering that the fixing of variables $\alpha_{j,i}$ according to (3) can be done during preprocessing.

When modeling the unrestricted case we must take into account that neither the height of each stack is necessarily given by its highest item, nor the highest stack of a stripe necessarily contains the stripe's highest item. Therefore, we additionally need variables $\beta_{k,j}$ for $j < k$, thus

- $\beta_{k,j} \in \{0,1\}$, $k = 1,\dots,n,\ j = 1,\dots,n$:
  set to one iff stack j is contained in stripe k.

To extend the ILP (1) to (8), we must in particular replace the height constraints for sheets (8). A straightforward way is to write

$$\sum_{k=l}^{n} \mathcal{H}(k)\gamma_{l,k} \leq\ H\gamma_{l,l}, \quad \forall l = 1,\dots,n-1 \tag{12}$$

with $\mathcal{H}(k)$ being the height of stack k

$$\mathcal{H}(k) = \sum_{i=1}^{n} h_i \alpha_{k,i}. \tag{13}$$

The left hand sides of inequalities (12) are, however, nonlinear. In order to obtain an ILP, we introduce additional variables

- $\delta_{l,i,j} \in \{0,1\}$, $l = 1,\dots,n-1$, $i = l+1,\dots,n$, and $j = l,\dots,i-1$:
  set to one iff item i contributes to the total height of all stripes in bin l and is contained in stack j; i.e., item i is contained in stack j, stack j is contained in stripe j (and therefore defines its height), and stripe j is contained in bin l or simply

$$\delta_{l,i,j} = 1 \ \leftrightarrow\ \alpha_{j,i} = 1 \wedge \gamma_{l,j} = 1. \tag{14}$$

Now, we can calculate the used height of a bin l in a linear way by

$$\sum_{i=l}^{n} h_i \gamma_{l,i} + \sum_{i=l+1}^{n} h_i \sum_{j=l}^{i-1} \delta_{l,i,j}. \tag{15}$$

The complete ILP for the (unrestricted) 3-stage 2BP looks as follows.

$$\text{minimize} \quad \sum_{l=1}^{n} \gamma_{l,l} \tag{16}$$

$$\text{subject to} \quad \sum_{j=1}^{i} \alpha_{j,i} = 1, \ \forall i = 1, \ldots, n \tag{17}$$

$$\sum_{i=j+1}^{n} \alpha_{j,i} \le (n-j)\alpha_{j,j}, \quad \forall j = 1, \ldots, n-1 \tag{18}$$

$$\alpha_{j,i} = 0, \ \forall j = 1, \ldots, n-1 \ \forall i > j \mid w_i \neq w_j \vee h_i + h_j > H \tag{19}$$

$$\sum_{k=1}^{n} \beta_{k,j} = \alpha_{j,j}, \quad \forall j = 1, \ldots, n \tag{20}$$

$$\sum_{i=j}^{n} h_i \alpha_{j,i} \ < \ \sum_{i=k}^{n} h_i \alpha_{k,i} + (H+1)(1-\beta_{k,j}), \quad \forall k = 2, \ldots, n, \quad \forall j = 1, \ldots, k-1 \tag{21}$$

$$\sum_{i=j}^{n} h_i \alpha_{j,i} \ \le \ \sum_{i=k}^{n} h_i \alpha_{k,i} + H(1-\beta_{k,j}), \quad \forall k = 1, \ldots, n-1, \quad \forall j = k+1, \ldots, n \tag{22}$$

$$\sum_{j=1}^{n} w_j \beta_{k,j} \ \le \ W\beta_{k,k}, \quad \forall k = 1, \ldots, n \tag{23}$$

$$\sum_{l=1}^{k} \gamma_{l,k} = \beta_{k,k}, \quad \forall k = 1, \ldots, n \tag{24}$$

$$\sum_{i=l}^{n} h_i \gamma_{l,i} + \sum_{i=l+1}^{n} h_i \sum_{j=l}^{i-1} \delta_{l,i,j} \ \le \ H\gamma_{l,l}, \quad \forall l = 1, \ldots, n-1 \tag{25}$$

$$\alpha_{j,i} + \gamma_{l,j} - 1 \le \delta_{l,i,j} \le (\alpha_{j,i} + \gamma_{l,j})/2, \quad \forall l = 1, \ldots, n-1, \ \forall i = l+1, \ldots, n, \ \forall j = l, \ldots, i-1 \tag{26}$$

$$\sum_{k=l+1}^{n} \gamma_{l,k} \le (n-l)\gamma_{l,l}, \quad \forall l = 1, \ldots, n-1 \tag{27}$$

$$\alpha_{j,i} \in \{0,1\}, \ j = 1, \ldots, n, \ i = j, \ldots, n \tag{28}$$

$$\beta_{k,j} \in \{0,1\}, \ k = 1, \ldots, n, \ j = 1, \ldots, n \tag{29}$$

$$\gamma_{l,k} \in \{0,1\}, \ l = 1, \ldots, n, \ k = l, \ldots, n \tag{30}$$

$$\delta_{l,i,j} \in \{0,1\}, \ l = 1, \ldots, n-1, \ i = l+1, \ldots, n \tag{31}$$

The objective function (16) and constraints (17), (19), and (24) remain unchanged from the restricted model. Constraints (20) and (23) are also adopted, but some limits had to be modified in order to consider the additional $\beta_{k,j}$ variables. The other constraints are either new or were substantially changed. Constraints (4) and (5) are replaced by (20), (21), and (22). Since it is not guaranteed anymore that items are only assigned to a used stack j, inequalities (18) are introduced. Constraints (21) and (22) ensure that the height of each stack j never exceeds the height of the stripe k it is contained in (equal to stack k's height). We split these constraints into "strictly less" and "less than or equal to" constraints in order to avoid ambiguities when stacks have identical heights: The highest stack with the smallest index always determines the index k of the stripe. Inequalities (25) replace the height constraints for bins (8) and use expression (15) with the new variables $\delta_{l,i,j}$ for calculating a bin's used height. Constraints (26) force variables $\delta_{l,i,j}$ to be set to their intended values according to the definition in (14). Finally, inequalities (27) ensure that no stripes are packed into an unused bin.

In total the ILP uses $O(n^3)$ variables and constraints, and is, according to our knowledge, the first polynomial-sized ILP for 3-stage 2BP.

5 A Column Generation Approach

As an alternative approach for 3-stage 2BP we propose a set covering formulation with column generation. The formulation is based on a Dantzig-Wolfe decomposition [6] of the ILP from the previous section and ideas from Gilmore and Gomory [8,21] and Pisinger and Sigurd [16,17]. For a recent survey about selected topics in column generation see Lübbecke and Desrosiers [22].

Dantzig-Wolfe decomposition is an effective technique for obtaining stronger models and reducing the symmetry of some LP models. It splits a suitable ILP model into a linear master problem and smaller subproblems. In our case, all the constraints restricting the assignment of items to stacks and the assignment of stripes to a bin form the detached subsystem. The constraints requiring that each item is packed once remain in the master problem, which is further reformulated into a set covering model.

5.1 The Set Covering Model for 3-Stage 2BP

The following set covering model can, in principle, be applied to any bin packing problem because specific geometric constraints concerning feasible layouts are encapsulated in the pricing (or column generation) sub-problem.

Let $\mathcal{P}$ be the set of all feasible packings of a single bin. The variable $x_p \in \{0,1\}$ indicates whether a packing $p \in \mathcal{P}$ appears in the solution ($x_p = 1$) or not ($x_p = 0$). For every item $i = 1,\ldots,n$ and every packing $p \in \mathcal{P}$, let the constant $A_i^p = 1$ iff packing p contains item i; otherwise $A_i^p = 0$. The set covering model for bin packing can now be stated as

$$\text{minimize} \quad \sum_{p\in\mathcal{P}} x_p \tag{32}$$

$$\text{subject to} \quad \sum_{p\in\mathcal{P}} x_p A_i^p \geq 1, \quad \forall i = 1,\ldots,n \tag{33}$$

$$x_p \in \{0,1\}, \quad \forall p \in \mathcal{P} \tag{34}$$

Due to inequalities (33), solutions with items appearing more than once are feasible. If items must appear exactly once as in the case of our 3-stage 2BP, duplicates can simply be removed from the solution without destroying its feasibility or optimality.

In general, $\mathcal{P}$ is too huge for explicitly considering all variables x_p, $p \in \mathcal{P}$. We therefore use delayed column generation to solve the linear programming (LP) relaxation of the set covering model, called the *master Problem* (MP). In this way, we do not explicitly consider the majority of the variables x_p. We start with a small set of initial patterns $\mathcal{P}' \subset \mathcal{P}$ taken from an initial feasible solution and solve the LP relaxation of the problem restricted to $\mathcal{P}'$, the so-called *restricted master problem* (RMP). Based on the obtained solution, we search for a new pattern and its corresponding variable, whose inclusion in $\mathcal{P}'$ might allow for a better solution of the RMP. This extended LP is resolved and the whole process repeated until no further improvements are possible, and, therefore, an exact solution of the MP is obtained.

The *reduced costs* of a packing $p \in \mathcal{P}$ are

$$c_p^\pi = 1 - \sum_{i=1}^{n} A_i^p \pi_i, \tag{35}$$

where π_i are the dual variables from the solution of the RMP. Only variables with negative reduced costs can improve the current solution of the RMP, leading us to the challenge of finding such a variable/pattern.

Branching (described in Sec. 6.2) becomes necessary, if no further variables with negative reduced costs can be determined (therefore, the MP is solved) and the difference between the solution value of the MP and the value of the so-far best integer solution is greater than or equal to one, i.e., the optimality gap is greater than the granularity of the objective function.

5.2 The Pricing Problem

The pricing problem consists of finding a packing p with negative reduced costs c_p^π. The specific characteristics and constraints of the bin packing problem substantially determine this problem. Here, we consider the pricing problem for 3-stage 2BP, which is a *three-stage two-dimensional knapsack packing* (2DKP) problem with respect to profits π_i corresponding to the dual variable values of the current RMP solution. This is a two-dimensional rectangular Single Knapsack Problem (SKP) as defined in the typology of Wäscher et al. [5]. Based on the ILP for 3-stage 2BP presented in Sec. 4.2, the pricing problem can be formulated as follows.

$$\text{maximize} \quad \sum_{i=1}^{n} \pi_i \sum_{j=1}^{i} \alpha_{j,i} \tag{36}$$

$$\text{subject to} \quad \sum_{j=1}^{i} \alpha_{j,i} \leq 1, \quad \forall i = 1, \ldots, n \tag{37}$$

$$\sum_{i=1}^{n} h_i \sum_{j=1}^{i} \delta_{i,j} \leq H \tag{38}$$

$$\alpha_{j,i} + \beta_{j,j} - 1 \leq \delta_{i,j} \leq \frac{\alpha_{j,i} + \beta_{j,j}}{2}, \quad \forall i = 1, \ldots, n, \ \forall j = 1, \ldots, i \tag{39}$$

and the constraints
(18), (19), (20), (21), (22), and (23).

Variables $\alpha_{j,i}$ and $\beta_{k,j}$ have the same meaning as in the ILP of Sec. 4.2; variables $\gamma_{l,k}$ are not needed anymore. The variables $\delta_{i,j} \in \{0, 1\}$, $i = 1, \ldots, n$, $j = 1, \ldots, i$, replace $\delta_{l,i,j}$ and are set to one iff item i contributes to the total height of all used stripes, i.e., iff item i appears in stack j and stack j appears in stripe j. The correct values for $\delta_{i,j}$ are enforced by constraints (39). Constraints (37) ensure that no item is packed more than once. Constraints (38) limit the total height of all stripes to H.

5.3 Stabilizing Column Generation

In column generation a near-optimal solution is usually reached relatively quickly, but the closer the optimum is approached, the smaller the progress per iteration becomes. The observation that dual variable values do not smoothly converge to their respective optima but strongly oscillate is regarded as a major efficiency issue [22]. In order to reduce this effect, we try to stabilize the column generation process. We apply a stabilization approach using dual-optimal inequalities as suggested by Ben Amor et al. [23] and adapt the dual *subset inequalities* (which are indeed dual optimal inequalities), introduced by Valério de Carvalho [24], to our needs.

5.3.1 Dual Subset Inequalities for General 2BP

We adapt the idea of subset inequalities for the cutting stock problem, as described by Ben Amor et al. [23], to the general 2BP. The master problem for 2BP (P_{2BP}) and its dual (D_{2BP}) are:

$$P_{\text{2BP}}\colon \ \min \sum_{p \in \mathcal{P}} x_p \qquad\qquad D_{\text{2BP}}\colon \ \max \sum_{i=1}^{n} \pi_i$$

$$\sum_{p \in \mathcal{P}} x_p A_i^p \geq 1, \quad \forall i = 1, \ldots, n, \qquad\qquad \sum_{i=1}^{n} \pi_i A_i^p \leq 1, \quad \forall p \in \mathcal{P}$$

$$x_p \geq 0, \quad \forall p \in \mathcal{P}. \qquad\qquad \pi_i \geq 0, \quad \forall i = 1, \ldots, n.$$

where $\mathcal{P}$ corresponds to the set of feasible patterns for a specific type of 2BP. Using this notation, we can now state the proposition defining the dual subset inequalities for 2BP.

Proposition 1 *For any item $i \in \{1, \ldots, n\}$ and subset $S \subset \{1, \ldots, n\}$, any optimal solution π^* to D_{2BP} satisfies*

$$\pi_i^* \geq \sum_{j \in S} \pi_j^*$$

if the items from S can be packed into a bin of size $h_i \times w_i$ and any pattern containing item i remains feasible when item i is replaced by an appropriate sub-pattern containing all items from S.

Proof We adapt the proof by contradiction for the cutting stock problem given in [23]: Let $(x^*,\ \pi^*)$ be a pair of primal-dual optimal solutions and assume $\pi_i^* < \sum_{j \in S} \pi_j^*$ for item i and subset S, but the items from S can be packed into a bin of size $h_i \times w_i$. We further assume that for any feasible pattern containing items $\{i\} \cup R$, there is a feasible pattern obtained by replacing i by S. These patterns define two dual constraints:

$$\pi_i^* + \sum_{j \in R} \pi_j^* \leq 1 \quad \text{and} \quad \sum_{j \in S} \pi_j^* + \sum_{j \in R} \pi_j^* \leq 1. \tag{40}$$

The assumption made above implies that

$$\pi_i^* + \sum_{j \in R} \pi_j^* < \sum_{j \in S} \pi_j^* + \sum_{j \in R} \pi_j^* \leq 1$$

and that the left constraint of (40) cannot be active at optimality. Applying the complementary slackness condition, all primal variables corresponding to patterns containing item i must be equal to 0: this is a contradiction to the feasibility and optimality of x^*. $\square$

This general form of subset inequalities can be used for nearly arbitrary variants of 2BP, since they only have different definitions of feasible packing patterns. In order to get dual constraints for these problems, fast preprocessing heuristics can be used to generate effective feasible subset inequalities.

5.3.2 Dual Constraints for 3-stage 2BP

In the previous section we devised a general form of subset inequalities which we now put in concrete form for the 3-stage 2BP problem. Every item i of a specific pattern can be replaced by a stack s of items having the same width as i and a height not exceeding the one of i without breaking the feasibility of the 3-stage pattern. We can therefore introduce the following dual constraints:

$$(\forall j \in S : w_i = w_j) \ \wedge \ (h_i \geq \sum_{j \in S} h_j) \quad \rightarrow \quad \pi_i^* \geq \sum_{j \in S} \pi_j^*, \qquad \forall i = 1, \ldots, n \tag{41}$$

Valério de Carvalho [24] keeps the number of actually used constraints in $O(n)$ by using only those with $|S| = 1$ and some with $|S| = 2$. Using (41), we can devise the following *type-1* constraints for $|S| = 1$:

$$w_i = w_j \quad \rightarrow \quad \pi_i^* \geq \pi_j^*, \qquad \forall i = 1, \ldots, n-1, \ \forall j = i+1, \ldots, n \tag{42}$$

Type-2 constraints for $|S| = 2$ are devised similarly:

$$\begin{aligned} w_i = w_j = w_k \wedge h_i \geq h_j + h_k \quad &\rightarrow \quad \pi_i^* \geq \pi_j^* + \pi_k^* \\ \forall i = 1, \ldots, n-2, \forall j = i+1, \ldots, n-1, \forall k &= j+1, \ldots, n \end{aligned} \tag{43}$$

Since adding constraints (rows) to the dual problem corresponds to adding variables (columns) to the primal problem, the dual constraints induce columns for the master problem. For each constraint, a zero-cost variable $y_{i,S}$ with coefficients -1 in row i, $+1$ in rows $j \in S$, and 0 otherwise is introduced. This variable can be interpreted as indicator for a substitution of item i with the items of S in any feasible pattern containing i. When a pattern contains item i and the corresponding column $y_{i,S}$, the pattern in which i is replaced by S is implicitly considered and therefore has not to be created by column generation. Reconstructing a primal optimal solution from a solution and additional variables $y_{i,S}$ is straightforward [23]: for any overcovering item i (i.e., i appears more than once), the overcovering patterns must be modified by replacing item i with the items of set S for which $y_{i,S} > 0$.

6 The Branch-and-Price Framework

In order to get optimal integer solutions, we use a branch-and-price (B&P) framework. A general introduction to branch-and-price, including a discussion of branching schemes, is given in Barnhart et al. [25]. In the following, we present how to generate an initial feasible solution. Then we explain branching, which is in general necessary to solve the problem to integrality. Finally, we describe our hierarchical approach for solving the pricing problem.

```
Algorithm FFF($list$)
while $list$ not empty
  $b$ = new empty bin
  do
    $s$ = new empty stripe in bin $b$
    forall items $i$ in $list$
      if $s == \emptyset \wedge i.h \leq b.uh$
        initialize $s$ with $i$
        remove $i$ from $list$
      else if $s \neq \emptyset \wedge i.w \leq s.uw \wedge i.h \leq s.h$
        add $i$ to $s$
        remove $i$ from $list$
        forall items $j$ with $j > i$
          if $j.w == i.w \wedge i.h \leq s.uh$
            stack item $j$ on $i$
            remove $j$ from $list$
  while $s \neq \emptyset$ /* no more item fitted into bin */
  discard empty stripe $s$
  add $b$ to solution
```

Abbreviations:
$*.h$: height of *
$*.w$: width of *
$*.uh$: unused height of *
$*.uw$: unused width of *

Fig. 2. Finite first fit (FFF) for 3-stage 2BP.

6.1 Generating an Initial Feasible Solution

In order to initialize the column generation algorithm, a feasible solution is needed. The packing patterns of its bins are used as initial $\mathcal{P}'$. A promising feasible solution can often be derived by considering the restricted 3-stage 2BP model from Sec. 4.1 and trying to solve it using an ILP-solver with a given time limit. Another way of generating a feasible solution for 3-stage 2BP is the following order-based *finite first fit heuristic* (FFF). The heuristic gets an ordered list of all items as input. Its pseudocode is given in Fig. 2.

As long as the ordered list of items is not empty, the algorithm fills one bin after the other. Variable b represents the current bin and variable s the current stripe. Stacks of items are added to the current stripe as long as possible. If no further item can be placed in the current stripe, a new empty stripe is added to the current bin. If no items fit into this stripe, it is discarded, bin b is closed, and a new bin is initialized. The described algorithm initializes the stripes with stacks containing a single item and therefore generates restricted 3-stage 2BP solutions only. The solutions obtained by FFF strongly depend on the input order of the items. Therefore, multiple runs using different orders are often meaningful.

The outer two loops run together in time $O(n)$, since either an item is placed in the inner loop, which reduces the size of the item list, or a new bin is created.

The two inner loops have together a worst-case run-time of $O(n^2)$. The total worst-case effort of FFF for 3-stage 2BP is therefore $O(n^3)$.

6.2 Branching

If no further variables with negative reduced costs can be found by completely solving the pricing problem and the difference between the solution value of the RMP and the value of the so-far best integer solution exceeds one, branching becomes necessary. We use a branching rule similar to the one described in [16,17]. The solution space is divided into two parts, where two different items i_1 and i_2 have to be in different bins or in the same bin, respectively. We always choose the two highest items appearing in a pattern p whose corresponding variable x_p has an LP solution value closest to 0.5. Preliminary tests showed that other combinations of items were usually not better; even the use of strong branching [26] in order to choose between different branching candidates generated by diverse strategies did not improve the obtained results.

The first branch corresponds to adding the constraint

$$\sum_{p \in \mathcal{P}} x_p A^p_{i_1} A^p_{i_2} = 0, \tag{44}$$

the second branch corresponds to adding the two constraints

$$\sum_{p \in \mathcal{P}} x_p A^p_{i_1} (1 - A^p_{i_2}) = 0 \qquad \text{and} \qquad \sum_{p \in \mathcal{P}} x_p (1 - A^p_{i_1}) A^p_{i_2} = 0. \tag{45}$$

In an actual implementation it is not necessary to explicitly add constraints (44) and (45) to the RMP; instead, the variables violating the constraints are simply fixed to zero.

Furthermore, the following constraints have to be added to the subsequent pricing problems in order to guarantee that patterns violating the branching constraints are not generated anymore. In the first branch

$$\sum_{j=1}^{i_1} \alpha_{j,i_1} + \sum_{j=1}^{i_2} \alpha_{j,i_2} \leq 1, \tag{46}$$

and in the second branch

$$\sum_{j=1}^{i_1} \alpha_{j,i_1} = \sum_{j=1}^{i_2} \alpha_{j,i_2}. \tag{47}$$

In the sequel, we call i_1 and i_2 *conflicting* if constraint (46) is active, and say that i_1 *induces* i_2 and vice-versa if equation (47) is active.

6.3 Generating Columns using a Greedy Heuristic

Oliveira and Ferreira [10] suggested to perform a fast column generation by first applying a heuristic to quickly obtain a promising pattern with negative, but not necessarily minimal, reduced costs. Only if this heuristic fails, the pricing problem is solved with an exact method. Such an approach can lead to a faster overall column generation, since significantly fewer calls of the usually much slower exact algorithm are needed in general.

For the 3-stage 2BP, we suggest to use a four level hierarchy of pricing algorithms. Each of these algorithms searches for a variable with negative reduced costs, and if it fails, the next algorithm from the hierarchy is applied to the pricing problem. In each of these pricing iterations, a single variable is generated. The first level of the hierarchy is the greedy *first fit heuristic respecting branching constraints* (FFBC) shown in Fig. 3.

FFBC considers the items in the order given by the parameter *list*. One item i after the other is packed into the first stack it fits. If the item does not fit into any existing stack, a new stack is created in the first stripe it fits. If no such stripe exists and there is enough space left in the bin, a new stack is created and packed into a new stripe. Otherwise the algorithm proceeds with the next item. If the addition of an item to a stack would increase the corresponding stripe's height, we check if enough vertical space is left in the bin and actually add the item with a probability of 50%. The constraints resulting from branching are handled as follows. At the beginning, we recursively look for items induced by the current item i. If i and any of the induced items stay in conflict with any other induced or already packed item (checkAndFindInduced(i, *ind*) returns false), none of these items is packed. Otherwise we immediately try to pack i and all the induced items returned by checkAndFindInduced(i, *ind*) in parameter *ind*. If this turns out to be impossible, we skip the whole chain of items.

The outer loop of FFBC is performed $O(n)$ times. Procedure checkAndFindInduced() can be implemented in time $O(c)$, where c denotes the number of existing branching constraints. If there are induced items and not all of them could be packed together with i, it is never tried to pack those items again. Therefore, the pack procedure is called only $O(n)$ times, at most once for each item in the list. The pack procedure runs in time $O(n)$, since there are at most $O(n)$ positions where an item can be placed. The worst-case total run-time of FFBC is therefore $O(n^2 + nc)$.

6.4 Generating Columns using an Evolutionary Algorithm

When the greedy FFBC heuristic fails in finding a pattern with negative reduced costs, we apply a more sophisticated metaheuristic as second-level pric-

```
Algorithm FFBC($list$, $bin$)
forall $i$ in $list$ with $\pi_i > 0$
  if checkAndFindInduced($i$, $ind$)
  $topack = \{i\} \cup ind$
  $bin' = bin$
  $list = list - topack$
  do
    choose $j \in topack$ at random
    $topack = topack - \{j\}$
    if not pack($bin$, $j$)
      $bin = bin'$
      $topack = \emptyset$
  while $topack \neq \emptyset$
return $bin$
```

```
Function pack($b$, $i$)
forall stripes $s$ in $b$
  forall stacks $a$ in $s$
    if $w_i == a.w$
      if $h_i + a.h \leq s.h$
        pack $i$ into $a$
        return true
      else if $a.h + h_i - s.h \leq b.uh$
        with a probability of 50%:
          pack $i$ into $a$
          return true
forall stripes $s$ in $b$
  if $w_i \leq s.uw \wedge h_i \leq s.h$
    create stack $a$ containing $i$
    pack $a$ into $s$
    return true
  else if $w_i \leq s.uw \wedge h_i - s.h \leq b.uh$
    with a probability of 50%:
      create stack $a$ containing $i$
      pack $a$ into $s$
      return true
if $h_i \leq b.uh$
  create stack $a$ containing $i$
  pack $a$ into new stripe $s$
  pack $s$ into $b$
  return true
return false
```

Abbreviations:
*.h: height of *
*.w: width of *
*.uh: unused height of *
*.uw: unused width of *

Fig. 3. First fit heuristic respecting branching constraints (FFBC).

ing strategy. We decided to use an Evolutionary Algorithm (EA) operating directly on stripes, stacks, and items. Filho and Lorena [27] already successfully used an EA to generate columns for approximately solving graph-coloring problems.

6.4.1 Structure of the Evolutionary Algorithm

We use a standard steady-state algorithm applying binary tournament selection with replacement and duplicate elimination, see e.g. [28,29]. In each iteration, one new candidate solution is created by recombination of selected parents, and mutation is applied with a certain probability. The new candidate solution always replaces the worst solution of the population if it is not identical to an already existing solution.

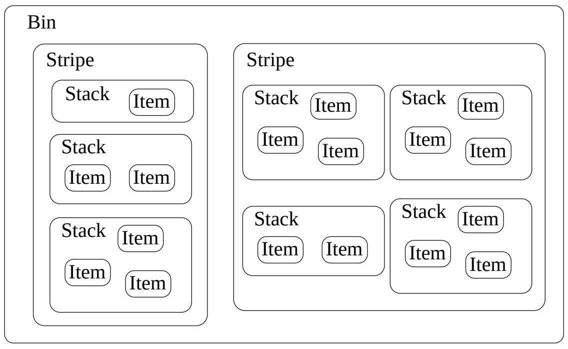

Fig. 4. Set-Representation of a solution (corresponding to the pattern from Fig. 1).

6.4.2 Representation and Initialization

The chosen representation is direct: each candidate solution represents a bin as a set of stripes, each stripe as a set of stacks, and each stack as a set of item references, see Fig. 4. Using such a hierarchy of sets makes it easy to ignore the order of items, stacks, and stripes and therefore to avoid symmetries.

Initial solutions are created via the FFBC heuristic using randomly generated item orders as input. These orders are, however, created in a biased way by assigning each item i a random value $r_i \in [0, 1)$ and sorting the items according to decreasing $r_i\pi_i$. Only items with $\pi_i > 0$ are considered.

6.4.3 Recombination

The recombination operator, shown in Fig. 5, first calculates for each stripe the sum p_s of all contained items' values π_i and a random value $r_s \in (0, 1]$. Then, the stripes are sorted according to decreasing $r_s p_s$. Thus, we obtain a random order which is biased in a way so that stripes having higher total values are more likely to appear at the beginning. The stripes are then considered in this order and packed into the offspring's bin if they fit into it. Identical stripes of both parents appear twice in the ordered list but are considered only at their first appearance. When all stripes have been processed, repairing is usually necessary in order to guarantee feasibility. First, the bin is traversed in order to delete item duplicates. Then, the branching constraints are considered: items staying in conflict with others (check(i) returns true) are removed. Afterwards, we try to pack all induced items (returned by findInduced(i)); if this is not possible, the corresponding original items are also removed from the bin. Finally, FFBC is applied as local improvement operator to a list of items not yet present in the bin and having positive π_i. These items are sorted by assigning each item i a random value $r_i \in [0, 1)$ and sorting them according to decreasing $r_i\pi_i$.

The *forall*-loops are all traversed $O(n)$ times. Procedures check() and findInduced() run in $O(c)$ time, where c denotes the number of existing branching

```
**Algorithm** StripeCrossover($A$, $B$)
/* crossover */
**forall** stripes $s$ in $A$ and $B$
    $p_s = \sum_{i \in s} \pi_i$
    $r_s =$ random value $\in [0, 1)$
$list$ = sort stripes by decreasing $r_s p_s$
$b$ = new bin
**forall** stripes $s$ in $list$
    **if** $s.h \leq b.uh$
        copy $s$ into $b$
/* repairing */
**forall** items $i$ in $b$
    **if** $i$ appears in $b$ twice
        remove first $i$ from $b$
**forall** items $i$ in $b$
    **if** check($i$)
        remove $i$ from $b$
**forall** items $i$ in $b$
    $ind$ =findInduced($i$)
    **if** $ind \neq \emptyset$
        $b' = b$
        $topack = ind - b$
        **do**
            choose $j \in topack$ at random
            $topack = topack - \{j\}$
            **if not** pack($b$, $j$)
                $b = b'$
                remove $i$ and items in $ind$
                    from $b$
        **while** $topack \neq \emptyset$
/* local improvement*/
$ilist = \{i = 1, \ldots, n \mid i \notin b \wedge \pi_i > 0\}$
sort $ilist$ by decreasing $r_i \pi_i$
    with $r_i$ = random value $\in [0, 1)$
FFBC($ilist$, $b$)
**return** $b$
```

Abbreviations:
$*.h$: height of *
$*.uh$: unused height of *

Fig. 5. Stripe crossover.

constraints. Procedure pack() is again at most once called for each item. The total run-time of stripe crossover is therefore bounded above by $O(n^2 + nc)$.

6.4.4 Mutation

The mutation operator removes a randomly chosen item i from the bin. If the branching constraints induce other items for i, they are also deleted. Finally, FFBC is used as local improvement operator, applied to a list of remaining items constructed in the same way as described for stripe crossover.

The mutation operator has a worst-case run-time of $O(n^2 + nc)$, since FFBC dominates the effort.

6.5 Pricing by Solving the Restricted 3-Stage 2DKP

Analogously to the restricted 3-stage 2BP, we can define a restricted version of our pricing problem, which we call *restricted 3-stage 2DKP*: The highest stack of each stripe may consist of a single item only. Formulating this restricted pricing problem as an ILP and trying to solve it using an ILP-solver is another heuristic approach for solving the general pricing problem. We apply this as third-level heuristic with a given time limit when the previous heuristics have failed. The ILP for restricted 3-stage 2DKP can be devised from the model (1) to (8), similarly to the unrestricted case (see Sec. 5.2):

$$\begin{aligned}
\text{maximize} \quad & \sum_{i=1}^{n} \pi_i \sum_{j=1}^{i} \alpha_{j,i} && (48)\\
\text{subject to} \quad & \sum_{j=1}^{i} \alpha_{j,i} \leq 1, \quad \forall i = 1, \ldots, n && (49)\\
& \sum_{k=1}^{n} h_k \beta_{k,k} \leq \; H && (50)\\
& \text{and the constraints:} \\
& \; (3), (4), (5), \text{and } (6).
\end{aligned}$$

6.6 Exact Pricing Algorithm

If all pricing heuristics failed in finding a pattern with negative reduced costs, we use an ILP-solver in order to solve the ILP for the unrestricted 3-stage 2DKP from Sec. 5.2. The optimization process is terminated—like the previous pricing heuristics—as soon as a pattern with negative reduced costs is found. Otherwise, the ILP-solver performs until it is proven that no such pattern exists. In this case, the solution of the RMP is also optimal for the MP and represents a valid lower bound for 3-stage 2BP.

7 Computational Experiments

The algorithms presented were implemented using GNU C++ 3.3.1 and the open-source library COIN [30], in particular the COIN/Bcp framework for branch-and-cut-and-price algorithms and COIN/Clp as LP-solver. Furthermore, CPLEX 8.1 was used with default parameters as general purpose ILP-solver. All experiments were performed on a 2.8GHz Pentium 4 machine.

7.1 Settings and Parameters

A global time limit of 1 000s was given to each run for all the approaches we tested.

Initial feasible solutions are generated with the FFF heuristic described in Sec. 6.1. FFF is called for $20n$ different item orders and the best obtained solution is used as initial one. The first five item orders are determined by sorting the items according to decreasing height, width, area, $2h_i + w_i$, and $h_i + 2w_i$, respectively; all further orders are random permutations. For some algorithm variants, the restricted 3-stage 2BP model was additionally tried to be solved using CPLEX with a time limit of 200s, and the overall best solution is used as initial feasible solution.

For solving the pricing problem, FFBC (see Section 6.3) is applied up to 100 times using different item orders. The first five item orders are determined by sorting the items according to decreasing π_i, $\frac{\pi_i}{h_i \cdot w_i}$, $\frac{\pi_i}{h_i + w_i}$, $\frac{\pi_i}{h_i}$, and $\frac{\pi_i}{w_i}$, respectively; all further orders are random permutations.

The following EA settings were determined by preliminary experiments and turned out to be robust for many different problem instances: a population size of 100, binary tournament selection, a mutation probability of 0.75, and termination after 1 000 iterations without improvement of the so-far best solution or a total of 100 000 iterations.

If FFBC or the EA did not find a solution, CPLEX is applied to the restricted 3-stage 2DKP ILP model within a time limit of 100s. We denote this pricing method by *CPLEX(restricted 3-stage 2DKP)*. Finally, if still no variable with negative reduced costs could be devised, CPLEX is used to solve the unrestricted 3-stage 2DKP model, denoted by *CPLEX(3-stage 2DKP)*.

In addition, the RMP is solved to integrality every M-th iteration by using CPLEX, possibly providing a new incumbent solution. $M = 100$ turned out to be a reasonable choice. The RMP is further solved to optimality before branching, because preliminary experiments showed that further branching can be sometimes avoided when a better upper bound is available.

7.2 Computational Results

In order to evaluate the effectiveness of the different models and algorithms described, we compare the following approaches:

2LBP:
CPLEX applied to the ILP for 2-stage 2BP from [12]

R2BP:
CPLEX applied to the ILP for restricted 3-stage 2BP

2BP:
CPLEX applied to the ILP for 3-stage 2BP
initialization: FFF and R2BP

EA:
The evolutionary algorithm from [4] applied to 3-stage 2BP; the specific variant used was EAe OX3,RX according to the notation in this article.

GuillSig:
B&P with constraint programming applied to guillotineable 2BP
Results adopted from [17] (run-time limited to 3 600s)

BPNoR:
B&P applied to 3-stage 2BP
initialization: FFF
pricing: FFBC, CPLEX(3-stage 2DKP)

BP:
B&P applied to 3-stage 2BP
initialization: FFF and R2BP
pricing: FFBC, CPLEX(restricted 3-stage 2DKP), CPLEX(3-stage 2DKP)

BPStab:
B&P applied to 3-stage 2BP
initialization: FFF and R2BP
pricing: FFBC, CPLEX(restricted 3-stage 2DKP), CPLEX(3-stage 2DKP)
stabilization: type 1 and type 2 constraints

BPStabEA:
B&P applied to 3-stage 2BP
initialization: FFF and R2BP
pricing: FFBC, EA, CPLEX(restricted 3-stage 2DKP), CPLEX(3-stage 2DKP)
stabilization: type 1 and type 2 constraints

The instances used for the experiments are adopted from Lodi et al. [12] and Pisinger and Sigurd [16,17]. We use the class numbering from [16,17]. The first six instance classes have the following characteristics:

Class 1: h_i and w_i uniformly random in $[1, 10]$, $W = H = 10$.
Class 2: h_i and w_i uniformly random in $[1, 10]$, $W = H = 30$.
Class 3: h_i and w_i uniformly random in $[1, 35]$, $W = H = 40$.
Class 4: h_i and w_i uniformly random in $[1, 35]$, $W = H = 100$.
Class 5: h_i and w_i uniformly random in $[1, 100]$, $W = H = 100$.
Class 6: h_i and w_i uniformly random in $[1, 100]$, $W = H = 300$.

In the last four classes, $W = H = 100$ and four types of items are used:

Type 1: w_i uniformly random in $[\frac{2}{3}W, W]$, h_i uniformly random in $[1, \frac{1}{2}H]$
Type 2: w_i uniformly random in $[1, \frac{1}{2}W]$, h_i uniformly random in $[\frac{2}{3}H, H]$
Type 3: w_i uniformly random in $[\frac{1}{2}W, W]$, h_i uniformly random in $[\frac{1}{2}H, H]$
Type 4: w_i uniformly random in $[1, \frac{1}{2}W]$, h_i uniformly random in $[1, \frac{1}{2}H]$

The instances classes are:

Class 7: type 1 with prob. 0.7, types 2, 3, and 4 with prob. 0.1 each.
Class 8: type 2 with prob. 0.7, types 1, 3, and 4 with prob. 0.1 each.
Class 9: type 3 with prob. 0.7, types 1, 2, and 4 with prob. 0.1 each.
Class 10: type 4 with prob. 0.7, types 1, 2, and 3 with prob. 0.1 each.

Each class has five sub-classes having n = 20, 40, 60, 80, and 100 items, and 10 instances exist in each sub-class. We therefore have a set of 500 instances in total.

Table 1 shows average lower bounds obtained from the LP-relaxations of the ILP models. Lower bound L_0 is the continuous lower bound as defined in [12]:

$$L_0 = \left\lceil \frac{\sum_{i=1}^{n} w_i h_i}{WH} \right\rceil \tag{51}$$

On a few instances, some column generation approaches did not terminate within the time limit of 1 000s and, therefore, no valid lower bounds could be found. In these cases, the values from L_0 are adopted.

The 2LBP and the R2BP bounds dominate L_0. While the ILP for restricted 3-stage 2BP gives relatively good lower bounds compared to the other presented methods, the lower bound derived from the unrestricted 3-stage 2BP model is generally low. This comes mainly from the $O(n^3)$ $\delta_{l,i,j}$ variables and the relatively weak constraints (18) and (27). The lower bounds of the column generation models dominate the others. One can observe that adding CPLEX(restricted 3-stage 2DKP) as additional pricing heuristic slightly improves the lower bounds, since column generation could more often terminate within the 1 000s time limit.

Results of the whole computational experiments are presented in Table 2 and Table 3. For each sub-class, the average objective value of the finally best feasible solution ($\overline{z}$), the number of times the algorithm could prove optimality of a solution (*Opt*), and the average computation time ($\overline{t}[s]$) are presented (the *Opt* column is omitted for the EA, since it is a heuristic method). The total averages are also given for every algorithm.

In general, differences in the objective values of final solutions obtained by the different optimization approaches are relatively small. One reason is the granularity of the objective function values (objective values are always integer). Another reason is that we only compared rather sophisticated approaches, which usually either find optimal solutions or solutions needing only one more bin than the optimum.

Table 2 shows the results of the first four algorithms. In the case of 2-stage 2BP, CPLEX was able to solve 95.2% of the instances to provable optimality, requiring 14.75 bins and 62.58s on average. The results computed for restricted 3-stage 2BP are slightly better in the number of needed bins (14.68), but slightly worse considering the percentage of proven optimality (94.8%) and the average run-time (68.46s). The restricted 3-stage model yields a lower average number of needed bins for every class, except for class 9 where the same number of bins is needed for 2-stage, 3-stage, and general guillotineable 2BP (see column GuillSig). When CPLEX is directly applied on the unrestricted

Table 1
Lower bounds.

Class	n	L_0	2LBP	R2BP	2BP	BPNoR	BP	BPStab	BPStabEA
1	20	6.4	6.4	6.4	3.7	7.0	7.0	7.0	7.0
	40	12.0	12.1	12.0	6.3	13.4	13.4	13.4	13.4
	60	18.5	18.5	18.5	9.8	20.0	20.0	20.0	20.0
	80	25.3	25.3	25.3	13.1	27.5	27.5	27.5	27.5
	100	30.5	30.5	30.5	14.7	31.5	31.5	31.5	31.5
2	20	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
	40	1.9	2.0	1.9	1.0	1.9	1.9	1.9	1.9
	60	2.5	2.5	2.5	1.0	2.5	2.5	2.5	2.5
	80	3.1	3.1	3.1	1.0	3.1	3.1	3.1	3.1
	100	3.9	3.9	3.9	1.0	3.9	3.9	3.9	3.9
3	20	4.4	4.4	4.4	2.4	5.4	5.4	5.4	5.4
	40	8.2	8.4	8.2	3.7	9.7	9.7	9.7	9.7
	60	12.5	12.7	12.5	5.4	13.9	14.0	14.0	14.0
	80	17.3	17.3	17.3	7.1	18.9	19.2	19.0	19.2
	100	20.5	20.7	20.5	7.9	21.6	21.8	22.0	21.8
4	20	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
	40	1.9	1.9	1.9	1.0	1.9	1.9	1.9	1.9
	60	2.3	2.5	2.3	1.0	2.3	2.3	2.3	2.3
	80	3.0	3.1	3.0	1.0	3.0	3.0	3.0	3.0
	100	3.7	3.7	3.7	1.0	3.7	3.7	3.7	3.7
5	20	5.4	5.5	5.4	2.9	6.6	6.6	6.6	6.6
	40	10.1	10.4	10.4	5.2	12.3	12.3	12.3	12.3
	60	15.7	15.8	15.7	8.1	18.3	18.3	18.3	18.3
	80	21.5	21.6	21.5	10.8	24.7	24.7	24.7	24.7
	100	25.9	26.1	26.0	12.7	27.7	28.5	28.5	28.5
6	20	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
	40	1.5	1.8	1.7	1.0	1.5	1.5	1.5	1.5
	60	2.1	2.1	2.1	1.0	2.1	2.1	2.1	2.1
	80	3.0	3.0	3.0	1.0	3.0	3.0	3.0	3.0
	100	3.2	3.2	3.2	1.0	3.2	3.2	3.2	3.2
7	20	4.7	4.9	4.7	3.3	5.7	5.7	5.7	5.7
	40	9.7	9.9	9.7	5.9	11.5	11.5	11.5	11.5
	60	14.0	14.1	14.0	7.8	16.1	16.1	16.1	16.1
	80	19.7	20.0	19.7	10.5	23.2	23.2	23.2	23.2
	100	23.8	23.8	23.8	11.8	27.1	27.1	26.4	27.1
8	20	4.8	5.0	5.0	3.5	6.1	6.1	6.1	6.1
	40	9.6	9.8	9.8	7.0	11.4	11.4	11.4	11.4
	60	14.1	14.3	14.3	10.3	16.4	16.4	16.4	16.4
	80	19.5	19.7	19.6	13.7	22.6	22.6	22.6	22.6
	100	24.1	24.1	24.1	16.8	28.0	28.0	28.0	28.0
9	20	9.4	9.4	9.4	7.6	14.3	14.3	14.3	14.3
	40	18.0	18.2	18.1	14.7	27.8	27.8	27.8	27.8
	60	27.6	27.9	27.9	22.9	43.7	43.7	43.7	43.7
	80	37.1	37.3	37.2	29.4	57.7	57.7	57.7	57.7
	100	45.0	45.1	45.1	35.9	69.4	69.4	69.4	69.4
10	20	3.8	4.1	4.1	1.4	4.5	4.5	4.5	4.5
	40	6.9	7.1	7.1	2.3	7.6	7.6	7.6	7.6
	60	9.4	9.9	9.7	2.9	9.8	10.2	10.1	10.2
	80	12.2	12.3	12.2	3.4	12.2	12.2	12.2	12.3
	100	15.3	15.5	15.3	3.6	15.3	15.3	15.3	15.3
	Average	11.96	12.08	12.01	6.87	14.48	14.52	14.50	14.52

Table 2
Experimental results 1.

		2LBP			R2BP			2BP			EA		GuillSig		
Class	n	$\overline{z}$	*Opt*	$\overline{t}$ [s]	$\overline{z}$	*Opt*	$\overline{t}$ [s]	$\overline{z}$	*Opt*	$\overline{t}$ [s]	$\overline{z}$	$\overline{t}$ [s]	$\overline{z}$	*Opt*	$\overline{t}$ [s]
	20	7.3	10	0.0	7.2	10	0.0	7.2	10	0.0	7.3	6.009	7.1	10	2.5
	40	13.8	10	0.2	13.7	10	0.1	13.6	7	356.6	13.8	11.074	13.4	10	7.9
1	60	20.3	10	0.3	20.1	10	0.4	20.1	3	701.2	20.4	16.598	20.0	10	422.2
	80	27.7	10	0.6	27.5	10	0.3	27.5	0	999.5	27.9	22.744	27.6	9	394.0
	100	32.4	10	0.9	31.8	10	23.1	31.8	3	693.1	32.6	40.732	31.9	8	936.5
	20	1.0	10	0.0	1.0	10	0.0	1.0	10	0.0	1	7.043	1.0	10	0.8
	40	2.0	10	0.2	2.0	10	0.2	2.0	9	100.1	2	11.999	2.0	10	291.8
2	60	2.8	9	100.9	2.6	9	191.5	2.7	8	160.6	2.8	16.862	2.6	6	1460.2
	80	3.3	10	97.6	3.3	8	212.8	3.3	8	162.3	3.4	27.44	3.3	1	3241.8
	100	4.1	9	134.8	4.0	9	173.9	4.1	8	166.5	4.1	39.825	4.0	3	2529.3
	20	5.4	10	0.0	5.4	10	0.0	5.4	10	0.1	5.4	6.066	5.1	10	4.2
	40	9.8	10	0.1	9.8	10	0.2	9.7	10	14.3	9.8	11.359	9.4	9	383.6
3	60	14.0	10	2.4	14.0	10	2.7	14.0	7	454.6	14.5	21.586	13.9	9	699.2
	80	19.7	10	3.8	19.4	10	4.6	19.4	0	995.2	19.8	38.679	19.0	9	584.9
	100	22.8	9	134.4	22.8	9	135.7	22.9	0	952.0	23.4	57.031	22.4	7	1173.0
	20	1.0	10	0.0	1.0	10	0.0	1.0	10	0.0	1	8.522	1.0	10	0.0
	40	2.0	10	0.2	2.0	10	1.3	2.0	9	99.0	2	15.95	1.9	10	55.4
4	60	2.7	8	222.2	2.5	10	36.1	2.6	7	279.9	2.9	24.74	2.5	4	2198.8
	80	3.4	9	103.7	3.4	7	304.3	3.4	6	340.2	3.4	36.744	3.3	0	3600.0
	100	4.2	6	403.0	4.2	5	508.3	4.2	5	401.1	4.2	56.44	3.8	3	2557.4
	20	6.7	10	0.0	6.7	10	0.0	6.6	10	0.0	6.7	7.73	6.5	10	1.3
	40	12.3	10	0.2	12.3	10	0.2	12.3	10	2.1	12.3	11.936	11.9	10	14.4
5	60	18.3	10	0.4	18.3	10	0.4	18.3	10	12.4	18.4	21.688	18.0	10	343.7
	80	25.0	10	3.4	24.9	10	3.2	24.8	8	309.5	25.1	41.331	24.7	9	745.3
	100	28.8	10	66.0	28.7	10	155.4	28.8	1	836.9	29.4	65.481	28.2	7	1295.5
	20	1.0	10	0.0	1.0	10	0.0	1.0	10	0.0	1	9.771	1.0	10	0.1
	40	1.9	10	0.3	1.9	10	0.4	1.9	10	17.6	1.9	18.406	1.9	8	1271.8
6	60	2.3	9	100.6	2.2	10	14.3	2.2	9	99.9	2.3	29.071	2.2	6	1813.7
	80	3.0	10	1.8	3.0	10	3.5	3.0	10	0.4	3	44.845	3.0	6	1499.0
	100	3.7	7	304.2	3.7	7	315.8	3.7	5	430.6	3.6	64.404	3.4	0	3600.0
	20	5.7	10	0.0	5.7	10	0.0	5.7	10	0.0	5.7	5.647	5.5	10	0.8
	40	11.5	10	0.2	11.5	10	0.1	11.5	9	160.0	11.5	19.644	11.1	8	1080.8
7	60	16.2	10	3.7	16.1	10	1.8	16.1	1	997.5	16.3	29.827	15.8	7	1083.0
	80	23.3	10	5.3	23.2	10	2.8	23.2	0	996.8	23.3	49.448	23.2	2	2881.3
	100	27.6	9	117.5	27.4	8	276.2	27.5	0	923.2	27.6	93.332	27.2	7	1102.3
	20	6.1	10	0.0	6.1	10	0.0	6.1	10	0.0	6.1	5.918	5.8	10	13.9
	40	11.5	10	1.1	11.4	10	1.6	11.4	10	0.0	11.8	8.24	11.3	9	721.8
8	60	16.4	10	2.0	16.4	10	7.0	16.4	10	0.1	16.5	13.927	16.1	8	1443.6
	80	22.7	9	139.8	22.6	10	40.1	22.6	10	4.5	23.1	20.227	22.4	9	369.7
	100	28.2	9	140.7	28.2	9	142.7	28.2	9	128.1	28.5	30.917	27.9	5	1819.0
	20	14.3	10	0.0	14.3	10	0.0	14.3	10	0.0	14.3	11.982	14.3	10	0.1
	40	27.8	10	0.0	27.8	10	0.0	27.8	10	0.0	27.8	16.976	27.8	10	0.6
9	60	43.7	10	0.0	43.7	10	0.0	43.7	10	0.1	43.7	22.408	43.7	10	2.2
	80	57.7	10	0.0	57.7	10	0.1	57.7	10	0.2	57.7	26.919	57.7	10	5.3
	100	69.5	10	0.1	69.5	10	0.1	69.5	10	0.3	69.5	43.266	69.5	10	16.0
	20	4.5	10	0.0	4.5	10	0.0	4.5	10	0.0	4.5	6.243	4.2	10	8.5
	40	7.7	10	1.8	7.7	10	1.8	7.7	9	123.8	7.7	16.222	7.4	9	383.0
10	60	10.5	9	117.7	10.4	9	101.3	10.4	6	512.1	10.5	27.519	10.1	7	1139.1
	80	13.5	7	327.5	13.2	8	229.3	13.2	1	832.6	13.7	41.058	13.1	2	2921.7
	100	16.4	7	589.5	16.4	6	529.3	16.5	0	866.0	16.8	61.214	16.5	0	3600.0
	Average	14.75	9.52	62.58	14.68	9.48	68.46	14.69	7.16	282.62	14.84	26.86	14.53	7.54	994.42

Table 3
Experimental results 2.

		BPNoR			BP			BPStab			BPStabEA		
Class	n	$\overline{z}$	*Opt*	$\overline{t}$ [s]	$\overline{z}$	*Opt*	$\overline{t}$ [s]	$\overline{z}$	*Opt*	$\overline{t}$ [s]	$\overline{z}$	*Opt*	$\overline{t}$ [s]
	20	7.2	10	0.3	7.2	10	0.3	7.2	10	0.6	7.2	10	4.2
	40	13.6	8	202.8	13.6	8	202.8	13.6	8	201.1	13.6	8	201.5
1	60	20.1	9	201.5	20.1	9	119.7	20.1	9	113.4	20.1	9	112.6
	80	27.5	10	50.7	27.5	10	43.4	27.5	10	53.4	27.5	10	68.6
	100	32.2	5	623.6	31.7	8	244.9	31.7	8	244.2	31.7	8	236.9
	20	1.0	10	0.1	1.0	10	0.0	1.0	10	0.1	1.0	10	0.1
	40	2.0	9	100.4	2.0	9	100.4	2.0	9	100.4	2.0	9	100.5
2	60	2.8	7	301.4	2.7	8	207.3	2.7	8	207.1	2.7	8	207.1
	80	3.4	7	303.8	3.3	8	228.1	3.3	8	228.6	3.3	8	228.1
	100	4.1	8	210.1	4.1	8	239.5	4.1	8	240.0	4.1	8	239.7
	20	5.4	10	0.2	5.4	10	0.2	5.4	10	0.2	5.4	10	0.3
	40	9.7	10	34.8	9.7	10	12.8	9.7	10	11.4	9.7	10	6.1
3	60	14.0	9	237.7	14.0	10	63.7	14.0	10	51.8	14.0	10	45.2
	80	19.3	8	397.4	19.2	10	174.2	19.3	9	197.2	19.2	10	166.8
	100	23.2	3	781.7	22.8	4	669.4	22.5	5	661.1	22.5	4	651.4
	20	1.0	10	0.1	1.0	10	0.1	1.0	10	0.1	1.0	10	0.1
	40	2.0	9	100.4	2.0	9	100.4	2.0	9	100.4	2.0	9	100.5
4	60	2.7	6	401.0	2.6	7	339.4	2.6	7	338.8	2.6	7	339.2
	80	3.3	7	302.8	3.3	7	321.4	3.3	7	321.3	3.3	7	321.5
	100	4.0	7	306.7	4.0	7	353.1	4.0	7	352.6	4.0	7	352.8
	20	6.6	10	0.2	6.6	10	0.2	6.6	10	0.2	6.6	10	0.5
	40	12.3	10	3.3	12.3	10	3.3	12.3	10	2.5	12.3	10	3.0
5	60	18.3	10	17.7	18.3	10	10.6	18.3	10	9.7	18.3	10	10.2
	80	24.8	9	138.9	24.8	9	128.1	24.8	9	127.9	24.8	9	129.7
	100	28.8	5	587.4	28.7	9	364.0	28.7	9	335.8	28.7	9	326.8
	20	1.0	10	0.1	1.0	10	0.0	1.0	10	0.0	1.0	10	0.0
	40	1.9	6	400.5	1.9	6	400.5	1.9	6	400.5	1.9	6	400.9
6	60	2.3	8	201.0	2.2	9	118.3	2.2	9	118.4	2.2	9	118.2
	80	3.0	10	3.1	3.0	10	13.9	3.0	10	13.9	3.0	10	14.0
	100	3.6	6	405.2	3.6	6	431.6	3.6	6	431.5	3.6	6	431.6
	20	5.7	10	0.4	5.7	10	0.3	5.7	10	0.3	5.7	10	0.6
	40	11.5	10	4.5	11.5	10	4.1	11.5	10	3.9	11.5	10	4.8
7	60	16.1	10	28.0	16.1	10	25.5	16.1	10	20.6	16.1	10	22.9
	80	23.2	10	57.3	23.2	10	80.6	23.2	10	79.3	23.2	10	77.8
	100	27.1	10	302.7	27.1	10	349.9	27.1	8	448.0	27.1	10	305.5
	20	6.1	10	0.8	6.1	10	0.9	6.1	10	0.9	6.1	10	1.0
	40	11.4	10	9.7	11.4	10	6.9	11.4	10	6.1	11.4	10	6.7
8	60	16.4	10	17.6	16.4	10	39.3	16.4	10	30.9	16.4	10	30.0
	80	22.6	10	94.8	22.6	10	106.1	22.6	10	79.7	22.6	10	79.5
	100	28.1	9	334.1	28.2	8	332.6	28.1	9	215.2	28.1	9	215.5
	20	14.3	10	0.1	14.3	10	0.0	14.3	10	0.0	14.3	10	0.1
	40	27.8	10	0.4	27.8	10	0.2	27.8	10	0.2	27.8	10	0.3
9	60	43.7	10	1.4	43.7	10	0.8	43.7	10	0.7	43.7	10	0.9
	80	57.7	10	3.6	57.7	10	2.4	57.7	10	2.3	57.7	10	2.6
	100	69.5	10	8.0	69.5	10	6.4	69.5	10	6.5	69.5	10	7.1
	20	4.5	10	0.9	4.5	10	0.5	4.5	10	0.6	4.5	10	0.4
	40	7.7	9	152.5	7.7	9	111.0	7.7	9	111.2	7.7	9	109.6
10	60	10.8	2	846.3	10.4	8	488.4	10.4	7	462.0	10.4	8	461.6
	80	13.9	0	1 000.0	13.2	1	902.5	13.2	1	902.5	13.2	2	889.1
	100	16.9	0	1 000.0	16.6	0	1 000.0	16.4	0	1 000.0	16.4	0	1 000.0
	Average	14.72	8.32	203.56	14.67	8.74	167.00	14.66	8.70	164.71	14.65	8.78	160.68

3-stage 2BP model without initialization, no solutions were found for some instances. We therefore used the same initialization as for the B&P algorithms, namely FFF in combination with CPLEX applied to the restricted 3-stage 2BP model. The average number of needed bins has slightly increased (14.69); furthermore, an increase of the average run-time (282.62s) and a decrease in the number of proven optimal solutions (71.6%) can be observed. The EA described in [4] needs 14.84 bins on average, and solves the instances within an average run-time of 28.86 seconds. This is the fastest of the considered algorithms, but its solution quality is the worst. The B&P algorithm of Pisinger and Sigurd applied to general guillotineable 2BP [16,17] yields a lower average number of needed bins (14.53), but the number of proven optimal solutions is relatively small (75.4%); furthermore the algorithm was given a global time limit of 3 600s.

Table 3 shows the results of the four B&P algorithms. The columns BPNoR and BP document the importance and the effectiveness of the use of the restricted 3-stage model inside the B&P algorithm: the average number of needed bins decreased from 14.74 to 14.67, and the percentage of optimally solved instances increased from 83.2% to 87.4%. Adding the dual subset inequalities further improves the results regarding the total number of needed bins and the run-time. Finally, using the EA and the stabilization yields the best results, the average number of needed bins is 14.65 and 87.8% of the instances have been solved to proven optimality. The run-time is 160.68s on average, which is the fastest of our four B&P variants.

In Table 4, the total number of pricing problems solved in each class as well as their sums are given for the B&P approaches. Furthermore, the bar charts shown in Fig. 6 visualize how often— in relation to the total number of solved pricing problems—the different pricing algorithms were successful for each class. The origin of the charts were shifted to 0.5 because for almost all the variants, the greedy FFBC algorithm solved more than half of the pricing problems.

In Table 4, we can observe that, when using FFBC only (BPNoR), the number of solved pricing problems is lower than the one of BP where CPLEX(restricted 3-stage 2DKP) is used. Pricing using a more sophisticated heuristic, in this case exactly solving restricted 3-stage 2DKP, can therefore improve the overall results, see also Table 3. Furthermore, the stabilizing inequalities reduce the number of pricing problems to be solved and, therefore, achieve the goal of stabilizing the column generation process. Adding the EA for column generation does not significantly change the total number of solved pricing problems.

Table 4
Number of solved pricing problems.

	BPNoR	BP	BPStab	BPStabEA
Class 1	12 049	12 130	7 536	7 706
Class 2	7 366	9 465	2 980	3 381
Class 3	16 988	16 552	13 531	13 257
Class 4	12 785	17 451	13 821	14 342
Class 5	15 455	15 612	14 188	13 503
Class 6	15 175	15 397	14 429	14 586
Class 7	17 334	16 962	16 228	16 429
Class 8	15 081	14 815	11 067	10 742
Class 9	658	567	490	491
Class 10	26 863	35 047	31 752	32 194
Total	139 754	153 998	126 022	126 631

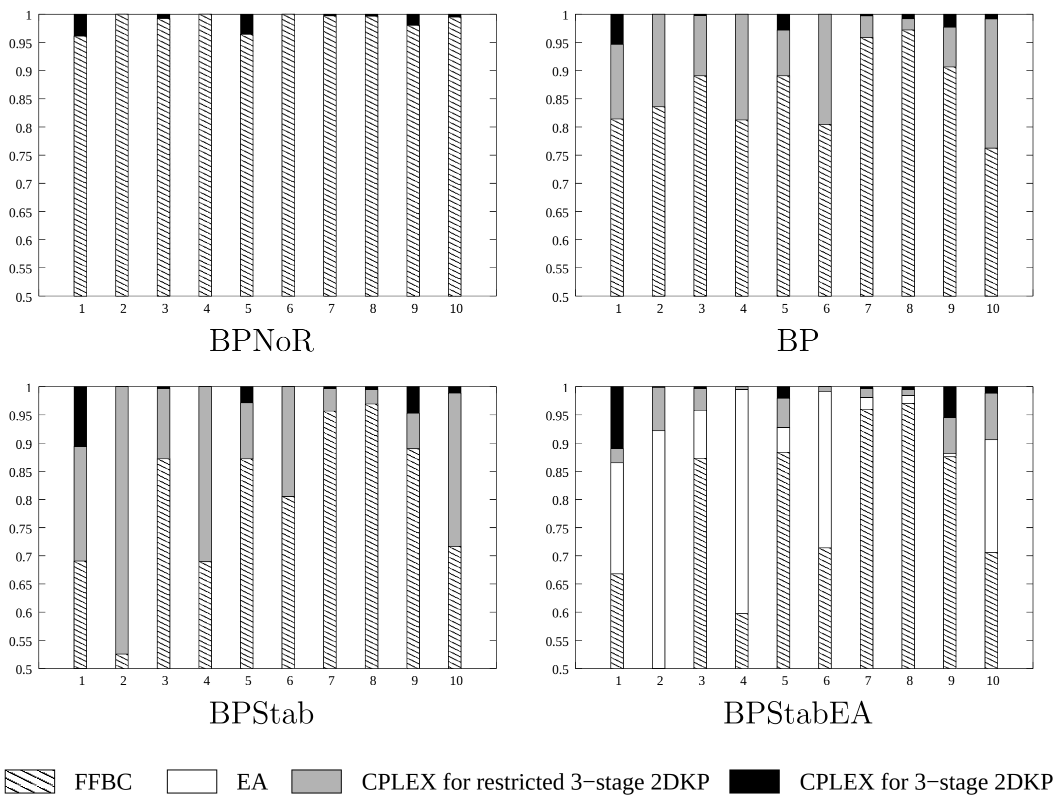

Fig. 6. Relative success rates of the four pricing algorithms.

The bar charts from Fig. 6 can be used to draw some conclusions about the characteristics of the pricing problems and the algorithms used to solve them. In general, we can observe that in every approach each implemented pricing algorithm solves a significant number of pricing problems; thus, no pricing algorithm is obviously obsolete. FFBC definitely solves the majority of pricing problems in all cases. These pricing problems can be denoted as “easy” ones. Looking at absolute numbers shows that CPLEX(restricted 3-stage 2DKP) successfully solved 21 500 pricing problems, which approximately corresponds to the increase of solved pricing problems when BPNoR is compared to BP in Table 4. The bar charts showing the relative success rates of the pricing algorithms indicate that the absolute number of “easy” pricing problems roughly remained the same. When stabilization is applied (BPStab), the total number of solved pricing problems is reduced; the absolute number of “easy” pricing problems decreased, whereas the absolute number of “harder” pricing problems remains approximately the same: CPLEX(restricted 3-stage 2DKP) successfully solved 22 751 pricing problems. Adding the EA (BPStabEA) does not significantly change the ratio of “easy” and “harder” pricing problems. A substantial number of those “harder” problems is solved by the EA (21 501), whereas CPLEX(restricted 3-stage 2DKP) is successful 4 915 times. Note that the total number of problems solved using CPLEX(3-stage 2DKP) is approximately the same for BP, BPStab, and BPStabEA: 1 602, 1 733, and 1 665, respectively.

8 Conclusions

We developed two polynomial-sized ILP models for 3-stage 2BP, a restricted model, and an unrestricted one. The restricted model is particularly useful for quickly obtaining near-optimal solutions to 3-stage 2BP. Solving the unrestricted model is computationally more expensive.

Further, a branch-and-price algorithm based on a set covering formulation for 2BP was proposed. This B&P algorithm was enhanced by dual subset inequalities stabilizing the column generation process. Column generation is performed by applying a hierarchy of up to four pricing methods having specific advantages and disadvantages: FFBC is the fastest, but can only solve “easy” pricing problems; the EA is slower, but is able to solve “harder” pricing problems in an efficient way. CPLEX(restricted 3-stage 2DKP) solves a restricted form of the pricing problem to proven optimality; finally, CPLEX(3-stage 2DKP) solves the unrestricted pricing problem to optimality, but is most time-consuming.

The ILPs for restricted and unrestricted 3-stage 2DKP were derived from the corresponding 3-stage 2BP models and proved to be efficient and very useful in the context of column generation.

We performed extensive computational experiments on standard benchmark instances in order to analyze the performance of the developed models and algorithms. The lower bounds obtained by column generation are strong. The best average results were achieved by B&P with all the proposed enhancements, in particular the four-level pricing strategy.

More generally, column generation performed by using a hierarchy of smart heuristics, also including meta-heuristics such as evolutionary algorithms and exact algorithms, can significantly improve the optimization speed and the capabilities of branch-and-price in finding optimal or near-optimal solutions.

Acknowledgements

We want to thank the anonymous referees for their helpful comments.

References

[1] A. Lodi, S. Martello, D. Vigo, Recent advances on two-dimensional bin packing problems, Discrete Applied Mathematics 123 (2002) 373–390.

[2] H. Dyckhoff, G. Scheithauer, J. Terno, Cutting and packing: An annotated bibliography, in: M. Dell'Amico, F. Maffioli, S. Martello (Eds.), Annotated Bibliographies in Combinatorial Optimization, Wiley, 1997, pp. 393–412.

[3] A. Fritsch, Verschnittoptimierung durch iteriertes Matching, Master's thesis, University of Osnabrück, Germany (1994).

[4] J. Puchinger, G. R. Raidl, G. Koller, Solving a real-world glass cutting problem, in: J. Gottlieb, G. R. Raidl (Eds.), Evolutionary Computation in Combinatorial Optimization – EvoCOP 2004, Vol. 3004 of LNCS, Springer, 2004, pp. 162–173.

[5] G. Wäscher, H. Haussner, H. Schumann, An improved typology of cutting and packing problems, European Journal of Operational Research this issue.

[6] F. Vanderbeck, On Dantzig-Wolfe decomposition in integer programming and ways to perform branching in a branch-and-price algorithm, Operations Research 48 (1) (2000) 111–128.

[7] J. O. Berkey, P. Y. Wang, Two-dimensional finite bin packing algorithms, Journal of the Operational Research Society 38 (1987) 423–429.

[8] P. C. Gilmore, R. E. Gomory, A linear programming approach to the cutting-stock problem (part I), Operations Research 9 (1961) 849–859.

[9] P. C. Gilmore, R. E. Gomory, Multistage cutting-stock problems of two and more dimensions, Operations Research 13 (1965) 90–120.

[10] J. F. Oliveira, J. S. Ferreira, A faster variant of the Gilmore and Gomory technique for cutting stock problems, JORBEL – Belgium Journal of Operations Research, Statistics and Computer Science 34 (1) (1994) 23 –38.

[11] M. Monaci, P. Toth, A set covering based heuristic approach for bin-packing problems, Tech. Rep. OR-03-1, Dipartimento di Elettronica, Informatica e Sistemistica, University of Bologna, under revision for INFORMS Journal on Computing (2003).

[12] A. Lodi, S. Martello, D. Vigo, Models and bounds for two-dimensional level packing problems, Journal of Combinatorial Optimization 8 (3) (2004) 363–379.

[13] F. Vanderbeck, A nested decomposition approach to a 3-stage 2-dimensional cutting stock problem, Management Science 47 (2) (1998) 864–879.

[14] E. Hopper, Two-dimensional packing utilising evolutionary algorithms and other meta-heuristic methods, Ph.D. thesis, University of Wales, Cardiff, U.K. (2000).

[15] S. Martello, D. Vigo, Exact solutions of the two-dimensional finite bin packing problem, Management Science 44 (1998) 388–399.

[16] D. Pisinger, M. Sigurd, Using decomposition techniques and constraint programming for solving the two-dimensional bin packing problem, Tech. Rep. 03/01, University of Copenhagen, Denmark, submitted for publication (2003).

[17] M. M. Sigurd, Column Generation Methods and Application, Ph.D. thesis, University of Copenhagen, Denmark (2004).

[18] A. Lodi, S. Martello, D. Vigo, Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems, INFORMS Journal on Computing 11 (1999) 345–357.

[19] J. Puchinger, G. R. Raidl, An evolutionary algorithm for column generation in integer programming: an effective approach for 2D bin packing, in: X. Yao et al. (Ed.), Parallel Problem Solving from Nature - PPSN VIII, Vol. 3242 of LNCS, Springer, 2004, pp. 642–651.

[20] G. Desaulniers, J. Desrosiers, M. M. Solomon, Accelerating strategies in column generation methods for vehicle routing and crew scheduling problems, in: C. C. Ribeiro, P. Hansen (Eds.), Essays and Surveys in Metaheuristics, Kluwer, Boston, 2001, pp. 309–324.

[21] P. C. Gilmore, R. E. Gomory, A linear programming approach to the cutting-stock problem (part II), Operations Research 11 (1963) 363–888.

[22] M. Lübbecke, J. Desrosiers, Selected topics in column generation, Les Cahiers du GERAD G-2002-64, HEC Montréal, Canada, under revision for *Operations Research.* (2002).

[23] H. Ben Amor, J. Desrosiers, J. Valério de Carvalho, Dual-optimal inequalities for stabilized column generation, Tech. Rep. G-2003-20, Les Cahiers du GERAD, HEC Montréal and GERAD, Canada, to appear in *Operations Research.* (2003).

[24] J. Valério de Carvalho, Using extra dual cuts to accelerate convergence in column generation, INFORMS Journal on Computing 17 (2) (2005) 175–182.

[25] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, P. H. Vance, Branch-and-price: Column generation for solving huge integer programs, Operations Research 46 (3) (1998) 316–329.

[26] A. Martin, General Mixed Integer Programming: Computational Issues for Branch-and-Cut Algorithms, in: M. Jünger, D. Naddef (Eds.), Computational Combinatorial Optimization: Optimal or Provably Near-Optimal Solutions, Vol. 2241 of LNCS, Springer, 2001, pp. 1–25.

[27] G. R. Filho, L. A. N. Lorena, Constructive genetic algorithm and column generation: an application to graph coloring, in: Proceedings of APORS 2000 - The Fifth Conference of the Association of Asian-Pacific Operations Research Societies within IFORS, 2000.

[28] T. Bäck, D. B. Fogel, Z. Michalewicz, Handbook of Evolutionary Computation, Oxford University Press, New York, 1997.

[29] Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs, Springer, Berlin, 1996.

[30] R. Lougee-Heimer, The Common Optimization INterface for Operations Research: Promoting open-source software in the operations research community, IBM Journal of Research and Development 47 (1) (2003) 57–66.