An Evolutionary Algorithm for Column Generation in Integer Programming: An Effective Approach for 2D Bin Packing

Jakob Puchinger, Günther R. Raidl

An Evolutionary Algorithm for Column Generation in Integer Programming: an Effective Approach for 2D Bin Packing

Jakob Puchinger and Günther R. Raidl

Institute of Computer Graphics and Algorithms
Vienna University of Technology, Vienna, Austria
{puchinger|raidl}@ads.tuwien.ac.at

Abstract. We consider the 3-stage two-dimensional bin packing problem, which occurs in real-world problems such as glass cutting. For it, we present a new integer linear programming formulation and a branch and price algorithm. Column generation is performed by applying either a greedy heuristic or an Evolutionary Algorithm (EA). Computational experiments show the benefits of the EA-based approach. The higher computational effort of the EA pays off in terms of better final solutions; furthermore more instances can be solved to provable 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 recent survey on 2D packing problems is given in Lodi et al. [5]. Among the algorithms for exactly solving the general 2BP problem are the branch and bound algorithm of Martello and Vigo [7] and the hybrid Branch and Price / Constraint Programming algorithm presented by Pisinger and Sigurd [8].

In many cases there is a special requirement on the cutting patterns: only orthogonal *guillotine* cuts are allowed, i.e., pieces may only be cut horizontally or vertically from one border to the one opposite. Furthermore, the number of stages of such cuts, i.e., the height of the cutting tree of each bin, is often limited in real-world applications. The case of two-stage cutting was first considered by Gilmore and Gomory [3]. More recently two-stage 2BP was considered in Lodi et al. [4] and Belov and Scheithauer [1]. Three-stage cutting problems were treated in Vanderbeck [10] and Puchinger et al. [9], where particular real-world problems with specific additional properties were considered.

In Sec. 2 we present an Integer Linear Programming (ILP) model for classical 3-stage 2BP, based on the model of [4]. In Sec. 3 a column generation formulation and a Branch and Price (B&P) framework, based on [8], are proposed. We describe a greedy heuristic in Sec. 4 and an evolutionary algorithm in Sec. 5 for solving the pricing problem within the B&P approach, i.e., for generating new columns. In Sec. 6 experimental results are given and analyzed.

This work is supported by the Austrian Science Fund (FWF) under grant P16263-N04.

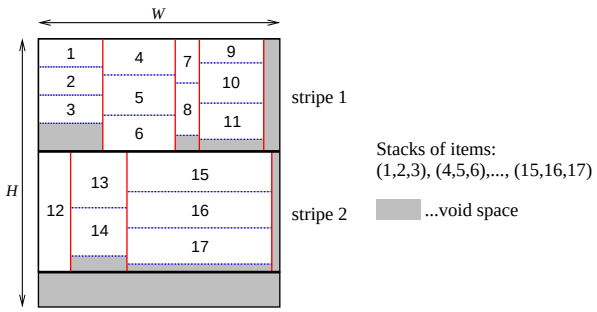

Fig. 1. A three-stage cutting pattern for one bin in normal form.

2 Three-Stage Two-Dimensional Bin Packing

The 2BP problem consists of a set of n rectangular items, each having a height h_i and a width 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.

A feasible layout for *3-stage* 2BP consists of a set of *bins*, each bin consists of a set of *stripes*, each stripe consists of a set of *stacks*, and each stack consists of *items* having equal width. Every such pattern can be reduced into its so-called *normal form* by moving each item to its uppermost and leftmost position, so that void space appears only at the bottom of stacks, to the right of the last stack in each stripe and below the last stripe, see Fig. 1. In the sequel we consider only patterns in normal form.

In [4] a polynomial-sized ILP model for 2-stage 2BP has been proposed. We extend this model in order to get a polynomial-sized ILP formulation for 3-stage 2BP.

The items are sorted so that $h_1 \geq h_2 \geq \ldots \geq h_n$. The order of the items within each stack is not relevant, so they can always be ordered according to their indices. A solution may contain at most n stacks. We label each stack with the index of the highest item it contains, i.e., the smallest item index. Similarly, a solution has at most n stripes, and a stripe's label is the label of its highest stack. Finally at most n bins are needed and we label each of them with the smallest index of the stripes it contains. The model uses the following 0/1-variables:

- $\alpha_{j,i}$, $j = 1, \ldots, n,\ i = j, \ldots, n$: rectangle i is contained in stack j;
- $\beta_{k,j}$, $k = 1, \ldots, n,\ j = 1, \ldots, n$: stack j is contained in stripe k;
- $\gamma_{l,k}$, $l = 1, \ldots, n,\ k = l, \ldots, n$: stripe k is contained in bin l;
- $\delta_{l,i,j}$, $l = 1, \ldots, n-1$, $i = l+1, \ldots, n$, and $j = l, \ldots, i-1$: item i contributes to the total height of all stripes in bin l; i.e., item i appears in stack j, stack j appears in stripe j, and stripe j appears in bin l.

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

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

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

$$\sum_{i=j+1}^{n} \alpha_{j,i} \leq (n-j)\alpha_{j,j}, \quad \forall j = 1, \ldots, n-1 \tag{3}$$

$$\alpha_{j,i} = 0, \ \forall j = 1, \ldots, n-1 \ \ \forall i > j \mid w_i \neq w_j \wedge h_i + h_j > H \tag{4}$$

$$\sum_{k=1}^{n} \beta_{k,j} = \alpha_{j,j} \quad \forall j = 1, \ldots, n \tag{5}$$

$$\sum_{i=j}^{n} h_i \alpha_{j,i} \ < \ \sum_{i=k}^{n} h_i \alpha_{k,i} + (H+1)(1-\beta_{k,j}),$$
$$\forall k = 2, \ldots, n, \quad \forall j = 1, \ldots, k-1 \tag{6}$$

$$\sum_{i=j}^{n} h_i \alpha_{j,i} \ \leq \ \sum_{i=k}^{n} h_i \alpha_{k,i} + H(1-\beta_{k,j}),$$
$$\forall k = 1, \ldots, n-1, \quad \forall j = k+1, \ldots, n \tag{7}$$

$$\sum_{j=1}^{n} w_j \beta_{k,j} \ \leq \ W\beta_{k,k}, \quad \forall k = 1, \ldots, n \tag{8}$$

$$\sum_{l=1}^{k} \gamma_{l,k} = \beta_{k,k}, \quad \forall k = 1, \ldots, n \tag{9}$$

$$\sum_{i=l}^{n} h_i \left(\gamma_{l,i} + \sum_{j=l}^{i-1} \delta_{l,i,j} \right) \ \leq \ H\gamma_{l,l}, \quad \forall l = 1, \ldots, n-1 \tag{10}$$

$$\alpha_{j,i} + \gamma_{l,j} - 1 \leq \delta_{l,i,j} \leq (\alpha_{j,i} + \gamma_{l,j})/2,$$
$$\forall l = 1, \ldots, n-1, \ \forall i = l+1, \ldots, n, \ \forall j = l, \ldots, i-1 \tag{11}$$

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

The objective function (1) minimizes the number of used bins. Equations (2) state that each item has to be packed once. In (3) it is ensured that items are only assigned to a *used* stack j, i.e., the stack contains item j. The fact that the items packed into the same stack must have identical width is guaranteed by (4). In an implementation, it is not necessary to use all of the $\alpha_{j,i}$, $j \neq i$, but only those for which $w_i = w_j$ and $h_i + h_j \leq H$. However, we keep them in our model for the sake of clarity. Each used stack is packed exactly once according to (5). Constraints (6) and (7) ensure for each stripe k and each contained

stack j that the stack's height does not exceed the stripe's height (= stack k's height). Constraints (8) guarantee that width W is not exceeded and no stacks are packed into unused stripes ($\beta_{k,k} = 0$). Equations (9) ensure each used stripe being packed into a bin. In (10) it is guaranteed that height H is not exceeded and no stripes are packed into unused bins ($\gamma_{l,l} = 0$). Constraints (11) force variables $\delta_{l,i,j}$ to be set to one iff rectangle i appears in stack j, stripe j is used, and stripe j appears in bin l. In (12) it is ensured that no stripes are packed into an unused bin.

3 Column Generation Formulation

A general introduction to integer linear programming, B&P, and column generation can be found in Wolsey [11]. Our column generation formulation for 3-stage 2BP is based on the set covering model from [8] and the ILP from the last section.

Let $\mathcal{P}$ be the set of all feasible 3-stage packings of a single bin. The 0/1-variable x_p indicates whether packing $p \in \mathcal{P}$ appears in the solution. For every rectangle $i = 1, \ldots, n$ and every packing $p \in \mathcal{P}$, let $A_i^p = 1$ iff packing p contains rectangle i; otherwise $A_i^p = 0$. The 3-stage 2BP problem can now be formulated as:

$$\text{minimize} \quad \sum_{p \in \mathcal{P}} x_p \tag{13}$$

$$\text{subject to} \quad \sum_{p \in \mathcal{P}} x_p A_i^p \geq 1 \qquad \forall i = 1, \ldots, n, \tag{14}$$

$$x_p \in \{0, 1\} \qquad \forall p \in \mathcal{P}. \tag{15}$$

In general, $\mathcal{P}$ is too huge for explicitly considering all variables x_p, $p \in P$. Fortunately, we can use delayed column generation to solve the Linear Programming (LP) relaxation of the problem without explicitly considering the majority of the variables [11]. 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 $\mathcal{P}'$. Based on the obtained solution, we search for a new variable/pattern whose inclusion in the restricted problem might improve the result. The extended LP is resolved and the whole process repeated until no further improvements are possible.

In addition, every M-th iteration the restricted problem is solved to integrality by using branch and cut, possibly providing a new incumbent solution ($M = 100$ turned out to be a reasonable choice).

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

$$c_p^\pi = 1 - \sum_{i=1}^{n} A_i^p \pi_i, \tag{16}$$

where π_i are the dual variables of the restricted LP-relaxed problem. Only variables with negative reduced costs can improve the current solution of the master problem leading us to the challenge of finding such a variable/pattern.

The Pricing Problem consists of finding a packing p with the smallest reduced costs c_p^π. It is a 3-Stage 2D Knapsack Packing (2DKP) problem with respect to the profits π_i. It can be modeled as follows:

$$\text{maximize} \quad \sum_{i=1}^{n} \pi_i \sum_{j=1}^{i} \alpha_{j,i} \tag{17}$$

$$\text{subject to} \quad \sum_{j=1}^{i} \alpha_{j,i} \leq 1, \quad \forall i = 1, \ldots, n \tag{18}$$

$$\sum_{i=1}^{n} h_i \sum_{j=1}^{i} \delta_{i,j} \leq H \tag{19}$$

$$\alpha_{j,i} + \beta_{j,j} - 1 \leq \delta_{i,j} \leq \frac{\alpha_{j,i} + \beta_{j,j}}{2} \quad \forall i = 1, \ldots, n,\ \forall j = 1, \ldots, i \tag{20}$$

and the constraints:
(3), (4), (5), (6), (7), and (8).

Variables $\alpha_{j,i}$ and $\beta_{k,j}$ have the same meaning as in the ILP of Sec. 2. The 0/1-variables $\delta_{i,j}$ 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.

Branching If no further variables with negative reduced costs can be determined, and the difference between the solution value of the LP-relaxed restricted problem and the value of the so-far best integer solution is greater than or equal to one, branching becomes necessary. We use a branching rule similar to the one described in [8]. The solution space is divided into two parts, where two different items i_1 and i_2 have not to be or have to be in the same bin. We always choose the two highest possible items from a pattern/variable with an LP solution value closest to 0.5.

The first branch corresponds to adding the constraint

$$\sum_{p \in \mathcal{P}} x_p A_{i_1}^p A_{i_2}^p = 0, \tag{21}$$

the second branch corresponds to adding the two constraints

$$\sum_{p \in \mathcal{P}} x_p A_{i_1}^p (1 - A_{i_2}^p) = 0 \qquad \text{and} \qquad \sum_{p \in \mathcal{P}} x_p (1 - A_{i_1}^p) A_{i_2}^p = 0. \tag{22}$$

In the actual implementation we do not explicitly add the constraints (21) and (22), but the variables violating them are fixed to zero.

The following constraints have to be added to the pricing problem in order to guarantee that patterns violating the branching constraints cannot be generated. In the first branch

$$\sum_{j=1}^{i_1} \alpha_{j,i_1} + \sum_{j=1}^{i_2} \alpha_{j,i_2} \leq 1, \tag{23}$$

and in the second branch

$$\sum_{j=1}^{i_1} \alpha_{j,i_1} = \sum_{j=1}^{i_2} \alpha_{j,i_2}. \tag{24}$$

In the sequel, we call i_1 and i_2 *conflicting* if constraint (23) is active and say that i_1 *induces* i_2 and vice-versa if equation (24) is active.

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 $\mathcal{P}'$. This solution is generated by the order-based Finite First Fit heuristic from [9] without considering the additional application-specific constraints described there. The heuristic is called $20n$ times for different item orders, and the best obtained solution is used. The first five item orders are determined by sorting the items according to decreasing height, width, area, $2h_i + w_i$, and $h_i + 2w_i$; all further orders are random permutations.

4 Solving the Pricing Problem

The pricing problem is solved by a greedy First Fit heuristic respecting the Branching Constraints (FFBC); see Fig. 2.

Similarly to the initialization heuristic, FFBC considers the items in a given order. One item 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 0.5.

The constraints resulting from branching are handled as follows. If an item is considered for packing, we first check whether there are any conflicts with already packed items in the bin (checkNoConflicts in Fig. 2); if there are, the item is skipped. Otherwise we recursively check if other items are induced, and if any of those stay in conflict with any other induced or already packed item (recursiveCheck in Fig. 2). If such a conflict occurs, none of these items can be packed. Otherwise, we immediately try to also pack all the induced items. If this is impossible we skip the whole chain of items.

FFBC is iteratively applied to up to 100 different item orders until a solution with negative reduced costs is found. The first five 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 $\frac{\pi_i}{w_i}$; all further orders are random permutations.

If the heuristic does not find a packing pattern with negative reduced costs, the general purpose ILP-solver CPLEX is finally called on the pricing problem in order to perform an exact search, eventually proving that no such patterns exist anymore.

```
Algorithm FFBC(items, bin)
forall items i with π_i > 0
  if checkNoConflicts(i)
    induced = recursiveCheck(i)
    if induced == ∅
      pack(bin, i)
    else
      tmp = bin
      packed = pack(tmp, i)
      if packed
        forall items j in induced
        packed = pack(tmp,j)
          if not packed
            break
      if packed
        bin = tmp

Abbreviations:
*.h: height of *
*.w: width of *
*.uh: unused height of *
*.uw: unused width of *
R: random value ∈ [0, 1)
```

```
Function pack(b, i)
forall stripes s in b
  forall stacks a in s
    if w_i == a.w
      if h_i + a.h ≤ s.h
        pack i into a
        return true
      else if a.h + h_i − s.h ≤ b.uh ∧ R < 1/2
        pack i into a
        return true
forall stripes r in b
  if w_i ≤ s.uw ∧ h_i ≤ s.h
    create stack containing i, pack it into s
    return true
  else if w_i ≤ s.uw ∧ h_i − s.h ≤ b.uh ∧ R < 1/2
    create stack containing i, pack it into s
    return true
if h_i ≤ b.uh
  create stack containing i
  pack it into new stripe, pack it into b
  return true
return false
```

Fig. 2. First fit heuristic respecting the branching constraints.

5 An Evolutionary Algorithm for the Pricing Problem

Since the 2DKP problem is strongly NP-hard, calling the ILP-solver may be very time-consuming. A more sophisticated metaheuristic, performed when FFBC did not find a variable with negative reduced costs and before solving the problem in an exact way, could lead to a faster overall column generation since significantly fewer calls of the ILP-solver may be needed. We decided to apply an Evolutionary Algorithm (EA) operating directly on stripes, stacks, and items.

Structure of the EA We use a standard steady-state algorithm with binary tournament selection and duplicate elimination. In each iteration, one new candidate solution is created by always applying recombination, and applying mutation with a certain probability. The new solution replaces the worst solution in the population if it is not identical to an already existing solution.

Representation and initialization The chosen representation is direct: Each chromosome represents a bin as a set of stripes, each stripe as a set of stacks, and each stack as a set of item references. Using such a hierarchy of sets makes it easy to ignore the order of items, stacks, and stripes and to avoid symmetries.

Initial solutions are created via the FFBC heuristic using randomly generated item orders. These orders are 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 $r_i \pi_i$.

Recombination This operator first assigns a random value $r_s \in [0, 1)$ to each stripe s in the two parent solutions. All these stripes are then sorted according to decreasing $r_s p_s$, with p_s being the sum of the π_i of the items contained in stripe s. The stripes are then considered in this order and packed into the offspring's bin when they fit into it (i.e., their height is smaller than the remaining unused height of the bin). Identical stripes of both parents appear twice in the ordered list, but they are considered at their first appearance only.

When all stripes have been processed, repairing is usually necessary in order to guarantee feasibility. First, the bin is traversed in order to check if items appear twice, the first of these items is deleted. Then, the branching constraints are considered: Items conflicting with others are removed. Afterwards, we try to pack induced items; if this is not possible the corresponding original items are also removed from the bin. Finally, FFBC is applied to the remaining items, possibly improving the solution.

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 applied to the remaining items for local improvement.

6 Experimental Results

We performed experiments on the benchmark instances from Berkey and Wang [2] (classes 1 to 6) and Martello and Vigo [7] (classes 7 to 10).

We compare CPLEX 8.1 directly applied to the ILP model (1) to (12) and the two variants of the B&P approach with and without the EA for solving the pricing problem. The B&P algorithm was implemented using the open-source framework COIN/Bcp (version 2004/04) [6], the LPs were solved using COIN/Clp. The computational experiments were performed on a Pentium 4 PC with 2.8 GHz. The EA's population size was 100, the mutation was performed with probability 0.75, and the EA terminated when either 1 000 iterations were performed without an improvement of the best solution or after a total of 100 000 iterations. Each of the experiments had a time limit of 1 000 seconds, which was occasionally exceeded because CPLEX is given the same time limit of 1 000 seconds.

Table 1 shows results obtained for the 10 problem classes; in each class there are 50 instances divided into 5 subclasses with $n = 20, \ldots, 100$ items. For each of the considered algorithms (CPLEX, B&P, B&P with EA), average objective values $\overline{z}$ of finally best integer solutions, numbers of instances solved to provable optimality *Opt* (out of 10), and average times $\overline{t}$ in seconds are given. The last rows show totals and averages over all instances.

When CPLEX is directly applied to the ILP model, 335 out of 500 instances could be solved to provable optimality. This is not bad, but substantially less than B&P's 402 instances and in particular the 409 completely solved instances of the EA-enhanced B&P. The differences in the objective values of the finally best integer solutions found by the three algorithms for instances that could not be

		CPLEX			B&P			B&P with EA		
Class	n	$\overline{z}$	*Opt*	$\overline{t}$ [s]	$\overline{z}$	*Opt*	$\overline{t}$ [s]	$\overline{z}$	*Opt*	$\overline{t}$ [s]
	20	7.2	10.0	0.0	7.2	10.0	0.3	7.2	10.0	6.3
	40	13.6	6.0	404.8	13.6	8.0	206.1	13.6	8.0	204.0
1	60	20.2	3.0	748.9	20.2	8.0	256.9	20.1	8.0	221.6
	80	27.8	0.0	1000.6	27.6	9.0	182.1	27.6	9.0	183.9
	100	32.7	0.0	1001.3	32.3	4.0	845.6	32.0	6.0	590.4
	20	1.0	10.0	0.0	1.0	10.0	0.1	1.0	10.0	0.1
	40	2.0	9.0	100.2	2.0	9.0	113.0	2.0	9.0	118.3
2	60	2.8	7.0	300.8	2.8	7.0	317.9	2.8	7.0	411.2
	80	3.4	7.0	302.4	3.4	7.0	377.9	3.4	7.0	410.1
	100	4.1	8.0	206.7	4.1	8.0	344.0	4.1	8.0	220.8
	20	5.4	10.0	0.0	5.4	10.0	0.2	5.4	10.0	0.3
	40	9.7	8.0	307.2	9.8	9.0	124.3	9.7	10.0	7.0
3	60	14.2	5.0	704.3	14.2	7.0	386.9	14.1	9.0	184.4
	80	20.3	0.0	1000.4	19.5	7.0	423.9	19.3	8.0	280.2
	100	23.9	0.0	1000.8	23.2	2.0	950.1	22.9	3.0	946.9
	20	1.0	10.0	0.0	1.0	10.0	0.1	1.0	10.0	0.1
	40	2.0	9.0	100.1	2.0	9.0	100.7	2.0	9.0	166.6
4	60	2.6	7.0	353.7	2.7	6.0	413.3	2.7	6.0	736.5
	80	3.3	7.0	300.9	3.3	7.0	402.3	3.3	7.0	303.3
	100	4.0	7.0	302.0	4.0	7.0	378.2	4.0	7.0	306.9
	20	6.6	10.0	0.0	6.6	10.0	0.2	6.6	10.0	0.8
	40	12.3	10.0	24.2	12.3	10.0	23.0	12.3	10.0	3.1
5	60	18.3	10.0	10.3	18.3	10.0	19.0	18.3	10.0	89.0
	80	25.0	5.0	530.1	24.8	9.0	199.4	24.8	9.0	213.7
	100	29.4	1.0	901.5	28.9	5.0	621.6	28.9	7.0	536.3
	20	1.0	10.0	0.0	1.0	10.0	0.1	1.0	10.0	0.1
	40	1.9	10.0	14.2	1.9	6.0	401.7	1.9	6.0	405.7
6	60	2.3	8.0	200.2	2.3	8.0	202.2	2.3	8.0	201.3
	80	3.0	10.0	0.4	3.0	10.0	3.1	3.0	10.0	3.1
	100	3.6	6.0	400.8	3.6	6.0	405.5	3.6	6.0	405.6
	20	5.7	10.0	0.0	5.7	10.0	0.3	5.7	10.0	0.7
	40	11.5	6.0	566.3	11.5	10.0	4.9	11.5	10.0	39.7
7	60	16.2	0.0	1000.2	16.2	9.0	128.5	16.1	10.0	24.7
	80	23.5	0.0	1000.4	23.2	10.0	60.2	23.3	9.0	157.3
	100	28.0	0.0	1000.7	27.1	10.0	292.1	27.1	10.0	269.6
	20	6.1	10.0	0.0	6.1	10.0	0.8	6.1	10.0	0.9
	40	11.4	10.0	0.6	11.5	9.0	133.7	11.4	10.0	120.1
8	60	16.4	10.0	8.6	16.5	9.0	116.0	16.5	9.0	118.6
	80	22.6	8.0	288.6	22.7	9.0	177.8	22.9	7.0	346.5
	100	28.2	8.0	410.0	28.3	7.0	506.4	28.4	6.0	508.9
	20	14.3	10.0	0.0	14.3	10.0	0.1	14.3	10.0	0.2
	40	27.8	10.0	0.0	27.8	10.0	0.4	27.8	10.0	0.6
9	60	43.7	10.0	0.1	43.7	10.0	1.4	43.7	10.0	1.5
	80	57.7	10.0	0.2	57.7	10.0	3.6	57.7	10.0	3.8
	100	69.5	8.0	200.3	69.5	10.0	7.9	69.5	10.0	8.5
	20	4.5	10.0	0.0	4.5	10.0	0.7	4.5	10.0	0.5
	40	7.7	9.0	185.8	7.7	9.0	166.1	7.8	8.0	217.4
10	60	10.7	3.0	822.1	10.7	2.0	857.2	10.5	3.0	825.1
	80	14.0	0.0	1000.4	13.9	0.0	1048.8	13.6	0.0	1114.5
	100	16.9	0.0	1000.7	16.9	0.0	1048.5	16.6	0.0	1151.4
Total		741.0	335.0	17702.0	737.5	402.0	12254.9	735.9	409.0	12068.2
Average		14.82	6.70	354.04	14.75	8.04	245.10	14.72	8.18	241.36

Table 1. Experimental results of the presented algorithms.

solved to optimality are in general relatively small. Nevertheless, B&P's solution values are in several cases significantly better than those of CPLEX, and the EA-enhanced B&P performs best on average. The two variants of B&P with and without the EA exhibit approximately the same total running times. Applying CPLEX directly was significantly slower in most cases. Thus, the application of the EA within the B&P framework is worth the additional effort.

7 Conclusions and Future Work

For 3-stage 2BP, we presented a compact ILP model having only $O(n^3)$ variables. In practice, however, the proposed column generation approach having a number of potential variables that grows exponentially with n turns out to be more efficient. Using the described EA as an additional strategy for solving the pricing problem pays off in terms of a higher capability of solving instances to provable optimality, but also slightly better average solution values. The combination of B&P and an EA in this form is also highly promising for other combinatorial optimization problems. Research on more sophisticated interaction and a parallel execution of these algorithms will be done next.

References

1. G. Belov and G. Scheithauer. A branch-and-cut-and-price algorithm for one-dimensional stock cutting and two-dimensional two-stage cutting. Technical Report MATH-NM-03-2003, Dresden University of Technology, Germany, 2003.
2. J. O. Berkey and P. Y. Wang. Two-dimensional finite bin packing algorithms. *Journal of the Operational Research Society*, 38:423–429, 1987.
3. P. C. Gilmore and R. E. Gomory. Multistage cutting-stock problems of two and more dimensions. *Operations Research*, 13:90–120, 1965.
4. A. Lodi, S. Martello, and D. Vigo. Models and bounds for two-dimensional level packing problems. *Journal of Combinatorial Optimization*. To appear.
5. A. Lodi, S. Martello, and D. Vigo. Recent advances on two-dimensional bin packing problems. *Discrete Applied Mathematics*, 123:373–390, 2002.
6. 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):57–66, 2003.
7. S. Martello and D. Vigo. Exact solutions of the two-dimensional finite bin packing problem. *Management Science*, 44:388–399, 1998.
8. D. Pisinger and M. Sigurd. Using decomposition techniques and constraint programming for solving the two-dimensional bin packing problem. Technical Report 03/01, University of Copenhagen, Denmark, 2003.
9. J. Puchinger, G. R. Raidl, and G. Koller. Solving a real-world glass cutting problem. In J. Gottlieb and G. R. Raidl, editors, *Evolutionary Computation in Combinatorial Optimization – EvoCOP 2004*, volume 3004 of *LNCS*, pages 162–173. Springer, 2004.
10. F. Vanderbeck. A nested decomposition approach to a 3-stage 2-dimensional cutting stock problem. *Management Science*, 47(2):864–879, 1998.
11. L. A. Wolsey. *Integer Programming*. Wiley-Interscience, 1998.