A Variable Neighborhood Search using Very Large Neighborhood Structures for the 3-Staged 2-Dimensional Cutting Stock Problem

Frederico Dusberger[1], Günther R. Raidl[1]

[1]Institute of Computer Graphics and Algorithms
Vienna Unviersity of Technology
Favoritenstr. 9/1861, 1040 Vienna, Austria
{dusberger|raidl}@ads.tuwien.ac.at

Abstract. In this work we consider the 3-staged 2-dimensional cutting stock problem, which appears in many real-world applications such as glass and wood cutting and various scheduling tasks. We suggest a variable neighborhood search (VNS) employing "ruin-and-recreate"-based very large neighborhood searches (VLNS). We further present a polynomial-sized integer linear programming model (ILP) for solving the subproblem of 2-staged 2-dimensional cutting with variable sheet sizes, which is exploited in an additional neighborhood search within the VNS. Both methods yield significantly better results on about half of the benchmark instances from literature than have been published before.

1 Introduction

Cutting and packing problems are among the most well-studied combinatorial optimization problems in literature. This is due to the versatility of these problems allowing many real-world applications to be modelled as such. In fact, both cutting and packing usually refer to one and the same problem, however it is common, according to the context, to use either one term or the other. Examples of applications include actual industrial glass, paper or steel cutting, container loading, VLSI design, or various scheduling tasks [1, 2]. Consequently, there are many different variants of the basic cutting and packing problems that have been discussed and for which a multitude of approaches already exists. Wäscher et al. [3] present an extensive typology of these problems, as well as a literature review of the most important works for the different variants. Nevertheless, this research area remains interesting, as additional modifications and side constraints arise, especially promoted due to new requirements from industry. Therefore, finding a most economical solution is still a challenging goal which calls for new approaches that are especially tailored and capable of respecting these new side constraints.

In this paper we consider in particular the 2-dimensional cutting stock problem (2CS), which – being a variant of the classical bin packing problem (1BP, or 2BP respectively) – is NP-hard [4]. In the basic problem setting one is given a set of rectangular elements which need to be cut from a minimal number of larger stock sheets. We also consider the common restriction that only *guillotine* cuts are allowed, i.e. cuts are always parallel to one of the sheet sides and reach from one border to the opposite one. A further common side constraint due to the restrictions of real-world cutting machines, is that the cutting has to be done using a certain number of *stages*. A k-staged cutting is a sequence of k stages of cuts, where each stage consists of a series of parallel guillotine cuts performed on the pieces obtained from the previous stage. The direction of the cuts in one stage is always orthogonal to the cuts in the previous stage. Here, we want to focus on 3-staged cutting, as this is a typical restriction, e.g. in the glass manufacturing industry [5, 6]. Secondly, experimental studies have shown that frequently no substantial gains can be obtained from more stages [7]. In case of the strip packing problem without allowing rotation the asymptotic performance ratio of a 3-staged cutting approximating an optimal cutting has been shown to be 1.69103 [8], whereas the 2-staged case is unbounded. The step from 2-staged to 3-staged cutting unfortunately dramatically increases the practical difficulty of the problem due to the possibility of stacking elements arbitrarily within a strip. In existing approaches there are often restrictions to the way elements can be stacked [9, 10], but often the solution quality can be significantly increased when dropping those restrictions.

In this work we present a variable neighborhood search (VNS) for the 3-staged 2CS which is composed by several very large neighborhood structures based on the "ruin-and-recreate" principle. Trying to further improve the solution quality we developed a new integer linear programming (ILP) model which also was applied in the reconstruction phase of a "ruin-and-recreate" neighborhood.

The next section provides a detailed problem description of the 2CS, followed by a literature review of the work related to our specific problem in section 3. In section 4 we present our VNS framework, with the basic large neighborhood searches. Section 5 describes the ILP model and how it is exploited in an additional large neighborhood search. Section 6 gives experimental results for the developed methods and an analysis thereof, and section 7 concludes this work.

2 Problem Definition

In the 2CS we are given a set of m rectangular *elements* $M = \{E_1, \ldots, E_m\}$ with dimensions $(h_1, w_1), \ldots, (h_m, w_m)$, also called *demand*, which can be grouped into $t \leq m$ *element types* having the same dimensions. Furthermore, we have a (potentially unlimited) stock of identical rectangular sheets of height $H > 0$ and width $W > 0$.

The objective is to find a *cutting pattern*, i.e. an arrangement of the elements in M on the stock sheets without overlap, s.t. the number of required sheets is minimal and the pattern can be cut in a 3-staged process. Elements are rotatable by 90°, which is reflected by adding for each element $E_i \in M$ an element E_{m+i} with $(h_{m+i}, w_{m+i}) = (w_i, h_i)$. We assume that an instance is generally feasible, i.e. $0 < h_i \leq H$ and $0 < w_i \leq W$ for each $E_i \in M$. Stage-1 and stage-3 cuts are always horizontal while stage-2 cuts are vertical.

We refer to the rectangles resulting from stage-1 cuts as *strips*, and the ones resulting from stage-2 cuts as *stacks*. We say that a cutting pattern is in *normal form*, if

(i) Waste occurs only at the bottom of stacks, at the right end of a strip and at the bottom of the sheet.
(ii) The topmost element of each stack is the widest one.
(iii) The leftmost stack of each strip is the highest one.

Clearly, every cutting pattern can be transformed into a pattern in normal form of equivalent quality. It is therefore sufficient to only consider patterns in normal form in the optimization.

We use the refined objective function proposed by Puchinger et al. [5] which considers the last sheet only partly. Let $S(x)$ denote the number of sheets used in a solution x and c_l the position of the last stage-1 cut of the last sheet. The objective function is then

$$f(x) = \min \left(S(x) - \frac{H - c_l}{H} \right), \tag{1}$$

This refinement allows for a more fine-grained distinction between solutions having an equal number of sheets. When considering real-world applications a cutting pattern yielding one larger waste area is typically preferred as this part can presumably more likely be reused than several smaller ones.

2.1 Solution Representation

We represent a solution explicitly by its *cutting tree* (in the literature also referred to as *slicing tree*). Note that a given cutting tree does not necessarily represent a complete solution, but can also be used as a representation of partial solutions or intermediate steps.

The root node represents the whole set of used sheets $S = \{s_1, \ldots, s_n\}$. Each of its children is associated with one sheet (and can be seen as a stage-0 cut). Every further level in the tree corresponds to a guillotine cut of the next stage l that has been applied. Note that the elements from M always appear as leaf nodes at the third level. Whenever an element actually is already finished after the stage-1 or stage-2 cuts, additional so-called *Null-cuts* are introduced to keep this consistent structure.
Each node N in the tree stores the dimensions (h, w) of the represented area, the waste within it and the absolute cut coordinate c on the sheet. If it is a leaf node, the associated element is stored. Figure 1 shows a 3-staged cutting pattern for a single sheet in normal form and the cutting tree representing it.

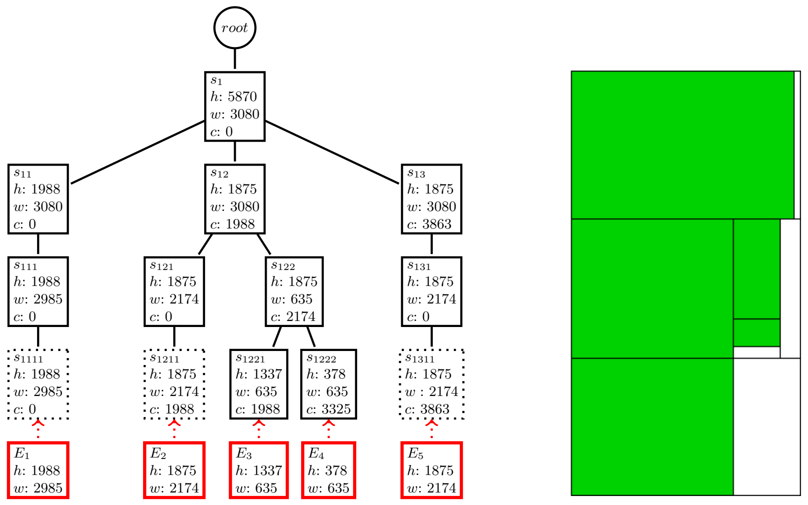
root
s_1 h: 5870 w: 3080 c: 0
s_{11} h: 1988 w: 3080 c: 0
s_{12} h: 1875 w: 3080 c: 1988
s_{13} h: 1875 w: 3080 c: 3863
s_{111} h: 1988 w: 2985 c: 0
s_{121} h: 1875 w: 2174 c: 0
s_{122} h: 1875 w: 635 c: 2174
s_{131} h: 1875 w: 2174 c: 0
s_{1111} h: 1988 w: 2985 c: 0
s_{1211} h: 1875 w: 2174 c: 1988
s_{1221} h: 1337 w: 635 c: 1988
s_{1222} h: 378 w: 635 c: 3325
s_{1311} h: 1875 w : 2174 c: 3863
E_1 h: 1988 w: 2985
E_2 h: 1875 w: 2174
E_3 h: 1337 w: 635
E_4 h: 378 w: 635
E_5 h: 1875 w: 2174

Fig. 1: A cutting tree (left) and the single sheet, s_1, represented by it (right). The leaf nodes $E_1, \ldots, E_5$ represent the actual elements obtained by the applied cutting pattern. The nodes s_{1111} and s_{1211} and s_{1311} are Null-cuts.

3 Related Work

The first exact solution approaches to the 2CS have been proposed by Gilmore and Gomory [11], who introduced the "exact two-stage guillotine cutting stock problem", which is a 2-staged 2CS under the additional constraint that all elements packed in one strip of the sheet have the same height. They further introduced the "non-exact two-stage guillotine cutting stock problem" where a final, third stage is allowed but only to separate an element from the waste area. Their approach is the well-known set covering formulation of the problem introducing a variable for each possible cutting pattern of a single sheet. Column generation is used to avoid the explicit enumeration of the exponentially many variables. Oliveira and Ferreira [12], already considering the 3-staged 2CS, presented a faster variant of this approach in which the pricing problem is solved by a greedy heuristic. The exact method is only applied in case the heuristic fails. A more recent approach is by Monaci and Toth [13] whose two-phase algorithm first creates cutting patterns using greedy heuristics which are then the columns in a set covering formulation solved by a Lagrangian-based heuristic algorithm. Alvarez-Valdes et al. [14] proposed a more sophisticated method for solving the pricing problem for the general n-staged 2CS without rotation using GRASP or tabu search for column generation. Puchinger and Raidl [10, 15] approached the 3-staged 2CS without rotation restricting the elements in a stack to be of equal width. They use a hierarchy consisting of a greedy construction heuristic, an evolutionary algorithm, a restricted ILP model for the pricing problem and an exact ILP model to generate columns, thus avoiding the computationally expensive uses of the exact method as much as possible.

An important step towards the reduction of the search space was done by Herz [16] who presented the concept of discretization points, excluding the vertical and horizontal coordinates at which no cut can occur in a pattern in normal form. This is also the first definition of so-called "canonical patterns" which correspond to patterns in normal form as defined in section 2. Later Christofides and Whitlock [17] showed a dynamic programming approach to compute them.

Since most of the exact approaches found in the literature are, in fact, hybrid approaches in one way or another, there are also numerous heuristic and metaheuristic approaches for the 2CS that have been studied. In [18] Lodi et al. give a survey on the concept and performance ratios for the prominent greedy heuristics, such as *first-fit*, *first-fit decreasing height* and *finite first-fit*.

4 Variable Neighborhood Search

The *Variable neighborhood search* (VNS) is a metaheuristic which relies on the idea of systematically searching for a better solution in an ordered set of increasingly complex neighborhood structures [19].

Frequently, VNS is combined with *very large(-scale) neighborhood search* (VLNS) techniques. The basic idea of a VLNS in contrast to a "classical" local search approach is to employ a special problem-specific large neighborhood structure $N(x)$, for which an efficient algorithm exists to derive an optimal or good approximate solution [20]. Methods that have been successfully used in VLNS for investigating neighborhoods include shortest path and matching algorithms, dynamic programming, (mixed) integer programming (MIP) and constraint programming. It is not always necessary to solve large neighborhoods to optimality but sometimes also simpler greedy constructive approaches can be applied. These approaches are often used in the context of so-called "ruin-and-recreate" methods, where one iteration consists of destroying randomly chosen or weak parts of an incumbent solution followed by a (usually relatively fast) recreation by a construction heuristic [21].
We follow these considerations in our VNS for the 3-staged 2CS, which is presented in the following.

4.1 Construction Heuristics

As a starting solution for the search the best among the solutions obtained by three different greedy construction heuristics is chosen. These approaches are *3-staged First Fit Decreasing Height with rotations* (3SFFDHR), 3SFFDHR preceded by a matching step (MATCH) and *Fill Strip* (FS), which are summarized in the following.

3SFFDHR This heuristic is based on the well-known first-fit decreasing height (FFDH) approach for the 2BP. The elements in M are ordered by non-increasing heights h_i; ties are broken randomly. The algorithm iterates through the element list trying to fit the current element in the cutting tree at the first possible position using a post-order traversal. At each level – given the dimensions of the parent and the already existing siblings – it is checked (i) if there is still enough room to accommodate the element, or (ii) if there is enough room for the rotated counterpart of the element. If so, a new sibling is created offering the required height (for a horizontal cut) or width (for a vertical cut) to accommodate the element.

In the worst case the algorithm needs to traverse trough the whole tree for inserting a new element. Therefore the runtime is in $\mathcal{O}(m^2)$.

MATCH The basic construction principle is the same as for 3SFFDHR. However, one of the disadvantages of the method is that the extent of each area created by a lower stage cut (height for stage-1 and width for stage-2) is fixed based on the element that initialized it.
To increase the space of potential stage-1 heights a preprocessing step is included based on the iterative matching approach by Fritsch and Vornberger [6].

A certain percentage p of the elements in M is chosen randomly and paired into meta-rectangles, s.t. the overall waste is minimal. This is done by first constructing a complete graph $G(V, E)$, where V corresponds to the elements and each edge $(u, v) \in E$ has associated a weight that is inversely proportional to the amount of waste in the bounding rectangle when aligning the elements corresponding to u and v. The meta-rectangles minimizing the overall waste are now determined by calculating a maximum weight matching on this graph.
The resulting meta-rectangles and the remaining elements that were not matched are then packed using the 3SFFDHR heuristic.

As the maximum-weight matching in a graph can be computed in $\mathcal{O}(|V|^3)$, the worst-case runtime of the MATCH construction heuristic is in $\mathcal{O}(m^3)$.

FS Fill Strip is a strip-based greedy construction heuristic and is an adaptation of the FFFWS heuristic proposed by Puchinger et al. [5]. As for 3SFFDHR the elements are ordered beforehand by non-decreasing height. The algorithm then basically fits the elements according to the same criteria they are fit with 3SFFDHR with the two following modifications:

– Whenever an element does not fit in the current subtree, it is skipped and the algorithm tries to fit the next one from M.
– Once the last element in M is reached, i.e. none of the remaining elements can fit in the current position in the cutting tree, the respective subtree is closed and never reconsidered again in the remaining construction process.
– When a strip is closed, the algorithm continues with the remaining unused elements in M beginning again with the highest one.

In the worst case the algorithm has to restart at the beginning of M for every new strip, hence its worst-case runtime is in $\mathcal{O}(m^2)$.

4.2 Neighborhood Structures and Search

All neighborhood structures we consider follow the principle of a ruin-and-recreate-based VLNS. We choose no “classical” local search neighborhood, since a local search approach based on solutions encoded by e.g. the order of the element list M is prone to suffer from poor locality. Provided no sophisticated decoding algorithm different from simple greedy construction heuristics is employed, a move as basic as a 2-exchange might lead to a rather different cutting pattern. Moreover, evaluating the objective of a solution requires decoding the cutting pattern completely, thereby forfeiting the runtime advantage a local search on a compact encoding usually comes with.

Maximum Waste Ratio We consider the maximum unused capacity in a used sheet as a secondary fitness value, following an idea that has also been successfully applied for the classical 1BP, cf. [22]. The waste ratio $wr(s)$ of a sheet s is defined as the free area on s (i.e. the area not covered by elements) relative to the total area of s. However, to be consistent with the (main) objective function the last sheet is also considered only partly when determining the maximum. Thus, we have

$$wr(s) = \begin{cases} \dfrac{c_l W - \sum_{i=1 \mid i \in elems(s)}^{m} h_i w_i}{c_l W}, & \text{if } s \text{ is the last sheet} \\ \dfrac{HW - \sum_{i=1 \mid i \in elems(s)}^{m} h_i w_i}{HW}, & \text{otherwise} \end{cases} \tag{2}$$

where $elems(s)$ is the set of elements that are placed on s and c_l defines the position of the last stage-1 cut of the sheet. The secondary fitness $G(x)$ of a solution x is therefore

$$G(x) = \max_{s \text{ non-empty}} wr(s) \tag{3}$$

$G(x)$ is used as a tie-breaking criterion for solutions with equal objective value to steer the search further towards more compact cutting patterns.

A basic step in our neighborhood search works as follows:
In the incumbent solution one or more subtree(s) of the cutting tree are removed, s.t. the elements associated with the leaves of these sub-trees become free. Using one of the aforementioned construction heuristics they are then reinserted again. Both steps can be efficiently done on the cutting tree representation. Next, we describe the variants for each the ruin and the recreate step in greater detail.

Ruin Subtree The general parameters for the ruin step are the tree-level λ and either a fixed number of subtrees of the level to be ruined (δ), or a percentage thereof (π). Furthermore, the range with respect to the affected sheets can be controlled. The subtrees may either be chosen randomly from all sheets or restricted to come from the same sheet(s), i.e. they are subtrees of the same level-0 node in the cutting tree. Independent from these settings the last sheet is always affected first, as both the objective function and the secondary fitness capture gradual improvements in terms of the last stage-1 cut of the last sheet. We consider two basic variants:

- *Ruin Random Subtree (RAND)*: Remove the defined number of level-λ subtrees from randomly selected positions in the tree respecting the sheet restrictions.
- *Ruin Max-Waste Subtree (MAX-W)*: Order the available level-λ subtrees by non-increasing waste ratio. Starting with the first one, remove the defined number from the tree respecting the sheet restrictions.

Recreate Cutting Tree After the ruin step the ruined subtree is normalized, i.e. the remaining subtrees are reordered (and the stored coordinates adapted), s.t. the pattern represented by the tree is in normal form. The removed elements are then sorted again by non-increasing height, shuffled or left unsorted in the order they were removed, before 3SFFDHR, MATCH or FS is used to reinsert them into the tree.

We also consider a next-improvement step function for the possible ruin and recreate combinations. In this work we use a fixed neighborhood order, which is shown in Table 1. Neighborhoods in which the ruin operation is restricted to affecting elements on the same sheet are marked as (f). The first five neighborhoods are chosen for intensification and finding improvements by relatively small changes. The following neighborhoods increasingly perturb the cutting tree by removing random and maximum waste subtrees and reinserting them in various ways. In the last neighborhoods, a special ruin operator is used, the removed subtrees are left intact, s.t. they can be reinserted as a whole (*Soft Remove*).

k	N_k-Ruin	N_k-Recreate	Step function
1	Random $(\lambda = 3, \delta = 2)$	3SFFDHR Unsorted	Next Imp.
2-3	Random $(\lambda = 2, \delta = k - 1)$	3SFFDHR Unsorted	Next Imp.
4-5	Random $(\lambda = 1, \delta = k - 3)$	3SFFDHR Unsorted	Next Imp.
6-9	Random $(\lambda = 3, \pi = (k - 5) \cdot 0.1)$	3SFFDHR Shuffled	Random
10-13	Random $(\lambda = 3, \pi = (k - 9) \cdot 0.1)$	MATCH Shuffled	Random
14-17	MAX-W $(\lambda = 2, \pi = (k - 13) \cdot 0.1)$	3SFFDHR Shuffled	Random
18-21	MAX-W $(\lambda = 2, \pi = (k - 17) \cdot 0.1)$	MATCH Shuffled	Random
22-25	MAX-W (f) $(\lambda = 2, \pi = (k - 21) \cdot 0.1)$	3SFFDHR Shuffled	Random
26-29	MAX-W (f) $(\lambda = 2, \pi = (k - 25) \cdot 0.1)$	MATCH Shuffled	Random
30-33	Soft $(\lambda = 1, \pi = (k - 29) \cdot 0.1)$	FFDH Shuffled	Random

Table 1: Neighborhoods and their order used in the VNS

5 ILP-Based Very Large Neighborhood Search

In the following we introduce a novel compact ILP model for the 2-staged 2CS with variable sheet size. This model is employed in the recreate step of a VLNS neighborhood. More particularly, it is used for optimally packing strips (i.e. defining the stage-2 and stage-3 cuts of the pattern). The remaining problem consists then of packing these strips into sheets.

5.1 An ILP for Packing Strips

Lodi et al. [23] proposed a polynomial-sized ILP model for the 2-staged 2CS. We base our ILP formulation on this model extending it to a model for optimally packing elements in strips of different sizes (in fact, it is a model for the 2-staged 2CS with variable sheet size). The dimensions of the different sheets reflect the dimensions of the stage-1 cuts that can then be placed on the actual sheets.

As defined in section 2 we denote by $t \leq m$ the number of different element types. Let further e_j be the number of elements of type $j \in \{1, \dots, t\}$ in the demand. The model is now based on the following considerations: We assume that m potential stacks are available. Each of them is associated with a different element a of a certain type i initializing it. Analogously, there are m potential strips, each initialized by a different potential stack a of type k (i.e. a stack initialized by an element of type k).
Furthermore, there are d different dimensions for the strips and each strip, as well as each stack, is of a certain type $l \in \{1, \dots, d\}$, defined by its dimensions.

We make use of the following observation which is similar to the definition of the normal form of a cutting pattern: For any optimal solution to the problem there exists an equivalent one in which the following conditions hold:

1. The first (topmost) element in each stack is the widest one in the stack.
2. The first (leftmost) element in each strip is the widest one in the strip.

We can further assume an ordering of the element types by nondecreasing width.

Finally, the occurrence of multiple elements of the same type in the demand can be exploited. In a given cutting pattern every permutation of such elements yields another pattern equivalent in structure and quality and can thus be considered symmetrical.

These considerations lead to the following 0/1-variables

$$y^l_{i_a} = \begin{cases} 1 & \text{if the } a\text{-th stack of dimension type } l \text{ is initialized} \\ & \text{by an element of type } i, \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

for $i = 1, \dots, t;\ l = 1, \dots, d;\ a = 1, \dots, m$

$$q^l_{k_a} = \begin{cases} 1 & \text{if the } a\text{-th strip of dimension type } l \text{ is initialized} \\ & \text{by a stack of type } k, \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

for $k = 1, \dots, t;\ l = 1, \dots, d;\ a = 1, \dots, m$
and the positive integer variables

- $x^l_{i_a j}$: The number of elements of type j packed in the a-th stack of dimension type l, initialized by an element of type i.
  for $i = 1, \ldots, t;\ j \geq i;\ l = 1, \ldots, d;\ a = 1, \ldots, m$
- $z^l_{k_a i}$: The number of stacks of type i packed into the a-th strip of dimension type l, initialized by a stack of type k.
  for $k = 1, \ldots, t;\ i \geq k;\ l = 1, \ldots, d;\ a = 1, \ldots, m$

Note that the variables inherently satisfy the guillotine cut restriction.
Our variant of the 2-staged 2CS with variable sheet size can now be stated as the following ILP:

$$\min \quad \sum_{l=1}^{d} \sum_{k=1}^{t} \sum_{a=1}^{m} q^l_{k_a} H_l \tag{6}$$

$$\text{s. t.} \quad \sum_{l=1}^{d} \sum_{a=1}^{m} \left(\sum_{i=1}^{j} x^l_{i_a j} + y^l_{j_a} \right) = e_j \qquad j = 1, \ldots, t \tag{7}$$

$$\sum_{j=i}^{m} h_j x^l_{i_a j} \leq (H_l - h_i) y^l_{i_a} \qquad i = 1, \ldots, t;\ a = 1, \ldots, m;\ l = 1, \ldots, d \tag{8}$$

$$\sum_{a=1}^{m} \left(\sum_{k=1}^{i} z^l_{k_a i} + q^l_{i_a} \right) = \sum_{a=1}^{m} y^l_{i_a} \qquad i = 1, \ldots, t;\ l = 1, \ldots, d \tag{9}$$

$$\sum_{i=k}^{m} w_i z^l_{k_a i} \leq (W_l - w_k) q^l_{k_a} \qquad k = 1, \ldots, t;\ a = 1, \ldots, m;\ l = 1, \ldots, d \tag{10}$$

$$\sum_{l=1}^{d} y^l_{i_a} \leq 1 \qquad i = 1, \ldots, t;\ a = 1, \ldots, m \tag{11}$$

$$\sum_{l=1}^{d} q^l_{k_a} \leq 1 \qquad k = 1, \ldots, t;\ a = 1, \ldots, m \tag{12}$$

The objective function (6) minimizes the total height of all used strips. Note that strip a of dimension type l is initialized by an element of type k, iff $q^l_{k_a} = 1$. Equations (7) ensure that each element j is packed exactly e_j times and constraints (8) impose that the height of each used stack does not exceed the respective dimension type's height. Analogously, equations (9) guarantee that each used stack is packed in a used strip while constraints (10) imply that the width of each used strip does not exceed the used dimension type's width. The last two groups of constraints strengthen the model by imposing that each potential stack (11) and each potential strip (12) a can only be used for one specific dimension type.

For the sake of clarity, we keep all variables in the model. However, in an actual implementation we can exclude the following variables, which can never be nonzero. For $1 \leq a \leq m$ and $1 \leq l \leq d$ we can set

$$y^l_{i_a} = 0 \qquad 1 \leq i \leq t \,|\, h_i > H_l \vee w_i > W_l \tag{13}$$

$$q^l_{k_a} = 0 \qquad 1 \leq k \leq t \,|\, h_k > H_l \vee w_k > W_l \tag{14}$$

$$x^l_{i_a j} = 0 \qquad 1 \leq i, j \leq t \,|\, h_i > H_l \vee w_i > W_l \vee h_j > H_l \vee w_j > W_l \vee h_i + h_j > H_l \tag{15}$$

$$z^l_{k_a i} = 0 \qquad 1 \leq k, i \leq t \,|\, h_i > H_l \vee w_i > W_l \vee h_j > H_l \vee w_j > W_l \vee w_i + w_j > W_l \tag{16}$$

Constraints (13) and (14) exclude variables for element types that do not fit in the respective dimension type, while (15) and (16) additionally rule out that two element types whose combination exceeds the dimensions can appear together. Finally, we can assume an ordering of the potential stacks and strips in accordance to the order of the element types, i.e.

$$a_1, \ldots, a_{e_1}, a_{(e_1+1)}, \ldots, a_{(e_1+e_2)} \ldots, a_{\sum_{i=1}^t e_i}$$

This reflects that each of the stacks (strips) is associated with exactly one element and we can additionally exclude the variables $y^l_{i_a}$ and $x^l_{i_a j}$ ($q^l_{k_a}$ and $z^l_{k_a i}$) according to the range associated with the element type i (k).

Note that the described ILP does not consider rotation of the elements. However, rotation can easily be incorporated by replacing the constraints (7) with

$$\sum_{l=1}^{d} \sum_{a=1}^{m} \left[\left(\sum_{i=1}^{j} x^l_{i_a j} + y^l_i \right) + \left(\sum_{i=1}^{\delta_j} x^l_{i_a \delta_j} + y^l_{\delta_j} \right) \right] = e_j, \;\; j = 1, \ldots, 2t; \; j < \delta_j,$$

where δ_j is the index of the rotated variant of type j in the element type order, and by replacing m with $2m$ and t with $2t$ in all the remaining equations, analogously to the variant proposed in [23].

In our ILP-VLNS approach we model the whole cutting tree but fix the variables corresponding to parts not selected by the ruin operator by the respective constants. The resulting strips obtained from solving the ILP are then inserted into the cutting tree, where completely new strips are packed using a FFDH strategy.

This approach is used in two variants within the VNS, see Table 2.

k	N_k-Ruin	N_k-Recreate	Step function
34	Random (f) ($\lambda = 3, \pi = 0.33$)	ILP	Random
35	Random (f) ($\lambda = 3, \delta = 0.33$)	ILP	Random

Table 2: Aditional neighborhoods in the ILP-based VNS

5.2 Determining Strip Dimensions

The remaining open question is how to determine the different strip dimensions, i.e. the different strip heights, that are considered by the model. Clearly, the already used dimensions in the parts of the cutting tree not affected by the ruin operator need to be included. For the additional dimensions there are, however, exponentially many possibilities in the number of element types, even when restricting the choices to the discretization points as proposed in [16].
In our approach this number is reduced during the ILP model generation by restricting the heights to multiples of the heights of available elements. In more detail, a random integer $n \in \{\lceil \frac{h_{\max}}{h_{\min}} \rceil, \ldots, \lceil \frac{H}{h_{\min}} \rceil\}$ is chosen, with $h_{\max}$ and $h_{\min}$ being the largest and the smallest height of all the element types. For each element type i the height dimensions $h_i \cdot k$, with $k \in \{1, \ldots, n\}$ are then generated. As this usually still yields a relatively large number of different heights, a quick evaluation is performed of how promising each of them is. This is done by first shuffling the list of free elements and then letting 3SFFDHR reinsert them into a strip of the given height. The strip heights that yield no more than the average waste ratio are finally chosen.

6 Experimental Results

Our algorithms have been implemented in C++, compiled with GCC version 4.6.3 and executed on a single core of a 3.40 GHz Intel Core i7-3770 with 16 GB RAM. For solving the developed ILP model we have used the general purpose MIP solver CPLEX version 12.6 with default parameter settings and a general time limit of 1000s, as well as a restriction to a single thread. Furthermore, the optimization was stopped as soon as an integer solution having a relative gap of less than or equal to 1% was found.

Computational experiments were performed on the benchmark instances from Berkey and Wang [24] (classes 1 to 6) and Martello and Vigo [25] (classes 7 to 10). Each class consists of 5 subclasses with $m = 20, \ldots, 100$ elements, each of which comprising 10 instances. We compare the basic VNS (VNS SIMPLE), the VNS consisting of the two ILP neighborhoods only (VNS ILP), a combination of both (VNS FULL) and the results obtained by the so far best-performing Branch & Price algorithm from [15] (BPStabEA), which were taken as presented in the respective work. In order to stay comparable to these results rotation of elements is not considered. For VNS SIMPLE, 25 major iterations over all neighborhoods are done, for VNS ILP and VNS FULL the VNS is stopped, when a major iteration does not yield an improvement. Each algorithm (VNS SIMPLE, VNS ILP, VNS FULL, BPStabEA) was applied five times to each problem instance and the average objective value and time spent per instance were determined. These values are then used for computing the average objective $\overline{f(x)}$ and time $\bar{t}$ for the instances of each subclass, which are shown in Table 3. The runtime for each of the experiments was additionally limited to 1000s. Occasionally, this limit was exceeded due to the same but independently measured time limit of 1000s given to CPLEX. The best objective value in each row is printed in bold. In the last

rows sums, average and median values over all instances are given.
In general, VNS SIMPLE outperforms VNS ILP and VNS FULL, whereas VNS FULL performs better than VNS ILP for all instance subclasses. We performed one-sided Wilcoxon signed rank tests comparing the objective for each instance class using a 95 % confidence interval. For five out of ten instance classes (2,4,6,9 and 10) both VNS SIMPLE and VNS FULL yield significantly better results than obtained by BPStabEA, VNS ILP is significantly better for classes 2,4,6 and 9. VNS FULL does not yield significantly better results than VNS SIMPLE but we observe an average increase in runtime. It can be expected that allowing VNS FULL to run for 25 major iterations would further improve the objective values, however, at the cost of a dramatic increase in the overall runtime.

7 Conclusions

We presented a VNS for the 3-staged 2-dimensional cutting stock problem which uses exclusively "ruin-and-recreate"-based VLNS. In a first straightforward approach greedy construction heuristics were used in the recreate step. We further developed a polynomial-sized ILP as an alternative method for recreating a solution. In fact, this model can also be applied for the 2-staged 2-dimensional cutting stock problem with variable sheet sizes. Experimental results on well-known problem instances show that the hybridization of VNS and VLNS indeed leads to a significant increase in solution quality for half of the instance classes. Using the ILP for recreation of the solution did not significantly increase the performance in comparison to the recreation by construction heuristics.

In future work, we want to improve the ILP-based VLNS in order to reduce the runtime allowing for more searches through larger neighborhoods. To this end we want to develop a more sophisticated approach to determine the dimensions for the strips and strengthen the model itself. Furthermore, it would be interesting to see, if the performance can be improved by inserting the strips with an exact method, e.g. an ILP formulation for the 1BP.

References

1. Lodi, A., Martello, S., Monaci, M.: Two-dimensional packing problems: A survey. European Journal of Operational Research **141**(2) (2002) 241–252
2. Murata, H., Fujiyoshi, K., Nakatake, S., Kajitani, Y.: VLSI module placement based on rectangle-packing by the sequence-pair. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on **15**(12) (1996) 1518–1524
3. Wäscher, G., Haußner, H., Schumann, H.: An improved typology of cutting and packing problems. European Journal of Operational Research **183**(3) (2007) 1109–1130
4. Garey, M.R., Johnson, D.S.: "Strong" NP-Completeness Results: Motivation, Examples, and Implications. Journal of the ACM **25**(3) (1978) 499–508
5. Puchinger, J., Raidl, G.R., Koller, G.: Solving a Real-World Glass Cutting Problem. In Gottlieb, J., Raidl, G.R., eds.: Evolutionary Computation in Combinatorial Optimization. Volume 3004 of Lecture Notes in Computer Science. Springer (2004) 165–176

		VNS SIMPLE		VNS ILP		VNS FULL		BPStabEA	
Class	m	$\overline{f(x)}$	$\overline{t}[s]$	$\overline{f(x)}$	$\overline{t}[s]$	$\overline{f(x)}$	$\overline{t}[s]$	$\overline{f(x)}^*$	$\overline{t}[s]^*$
	20	6.93	1.4	7.06	0.0	**6.90**	0.2	7.2	4.2
	40	**13.34**	10.4	13.47	0.2	13.40	1.4	13.6	201.5
1	60	**20.09**	33.2	20.34	1.0	20.13	5.6	20.1	112.6
	80	27.60	81.5	27.88	1.8	27.71	13.2	**27.5**	68.6
	100	32.24	161.4	32.31	3.2	32.25	22.1	**31.7**	236.9
	20	**0.74**	0.9	0.75	0.0	**0.74**	0.1	1.0	0.1
	40	**1.42**	6.7	1.44	0.5	**1.42**	1.6	2.0	100.5
2	60	**2.12**	22.8	2.14	1.9	**2.12**	4.7	2.7	207.1
	80	2.89	54.6	2.89	63.2	**2.87**	40.3	3.3	228.1
	100	**3.42**	113.9	3.43	89.3	3.43	86.3	4.1	239.7
	20	**4.90**	1.3	5.00	0.1	4.91	0.3	5.4	0.3
	40	**9.56**	9.7	9.83	0.4	9.65	1.7	9.7	6.1
3	60	14.16	33.0	14.46	1.9	14.31	6.1	**14.0**	45.2
	80	19.65	73.1	19.95	4.2	19.69	18.0	**19.2**	166.8
	100	23.23	141.8	23.53	8.0	23.24	44.9	**22.5**	651.4
	20	**0.75**	0.9	**0.75**	0.0	**0.75**	0.1	1.0	0.1
	40	**1.40**	5.8	1.41	2.1	**1.40**	3.8	2.0	100.5
4	60	**2.11**	20.9	2.12	22.7	2.12	21.4	2.6	339.2
	80	2.87	50.9	2.86	499.7	**2.85**	420.2	3.3	321.5
	100	**3.42**	100.2	**3.42**	532.1	**3.42**	618.3	4.0	352.8
	20	**6.30**	1.3	6.35	0.0	6.33	0.2	6.6	0.5
	40	**11.94**	10.1	12.16	0.5	11.97	2.4	12.3	3.0
5	60	18.35	35.1	18.63	2.1	18.46	10.9	**18.3**	10.2
	80	25.34	84.8	25.78	6.5	25.52	25.9	**24.8**	129.7
	100	29.38	164.3	29.75	14.6	29.49	55.9	**28.7**	326.8
	20	**0.66**	0.9	0.67	0.0	0.67	0.1	1.0	0.0
	40	**1.24**	6.4	1.25	1.2	**1.24**	2.4	1.9	400.9
6	60	**1.87**	21.2	**1.87**	45.9	**1.87**	70.2	2.2	118.2
	80	2.53	52.1	2.53	412.6	**2.52**	565.6	3.0	14.0
	100	3.03	107.5	3.03	748.5	**3.02**	818.3	3.6	431.6
	20	**5.16**	1.7	5.25	0.1	5.19	0.2	5.7	0.6
	40	**11.03**	11.7	11.12	0.5	11.05	2.4	11.5	4.8
7	60	**15.89**	42.3	16.09	2.0	15.95	9.6	16.1	22.9
	80	**22.79**	96.9	23.03	5.4	22.84	22.0	23.2	77.8
	100	27.11	185.2	27.26	12.2	27.16	47.4	**27.1**	305.5
	20	**5.69**	1.3	5.92	0.0	5.90	0.1	6.1	1.0
	40	11.42	10.2	11.56	0.7	11.45	1.9	**11.4**	6.7
8	60	16.24	30.9	16.48	6.8	**16.21**	22.2	16.4	30.0
	80	23.00	73.9	23.23	54.5	23.03	64.9	**22.6**	79.5
	100	28.28	143.0	28.40	92.5	28.20	190.6	**28.1**	215.5
	20	**13.86**	2.2	14.04	0.1	13.94	0.2	14.3	0.1
	40	**27.31**	15.8	27.61	0.7	27.49	1.8	27.8	0.3
9	60	**43.29**	55.7	43.64	1.7	43.49	5.6	43.7	0.9
	80	**57.21**	129.1	57.50	7.4	57.27	13.2	57.7	2.6
	100	**69.11**	242.8	69.43	13.3	69.23	29.3	69.5	7.1
	20	**3.96**	1.0	4.06	0.1	3.99	0.2	4.5	0.4
	40	**7.29**	7.9	7.42	0.6	7.33	1.9	7.7	109.6
10	60	**10.14**	24.3	10.41	3.0	10.15	10.1	10.4	461.6
	80	**13.03**	59.1	13.26	8.6	13.07	33.0	13.2	889.1
	100	**16.16**	127.1	16.37	50.4	16.17	100.3	16.4	1000.0
Sum		721.47	2670.0	729.12	2725.3	723.51	3419.2	732.7	8034.1
Average		14.43	53.4	14.58	54.5	14.47	68.4	14.65	160.68
Median		11.22	31.9	11.34	2.0	11.25	9.8	11.45	78.65

Table 3: Experimental results. The best objective value in each row is printed in bold. *Runtimes and objective values for BPStabEA are from [15].

6. Fritsch, A., Vornberger, O.: Cutting Stock by Iterated Matching. In Derigs, U., Bachem, A., Drexl, A., eds.: Operations Research Proceedings 1994. Volume 1994 of Operations Research Proceedings. Springer Berlin Heidelberg (1995) 92–97
7. Cintra, G., Miyazawa, F., Wakabayashi, Y., Xavier, E.: Algorithms for two-dimensional cutting stock and strip packing problems using dynamic programming and column generation. European Journal of Operational Research **191**(1) (2008) 61–85
8. Seiden, S.S., Woeginger, G.J.: The two-dimensional cutting stock problem revisited. Mathematical Programming **102**(3) (2005) 519–530
9. Vanderbeck, F.: A Nested Decomposition Approach to a Three-Stage, Two-Dimensional Cutting-Stock Problem. Management Science **47**(6) (2001) 864–879
10. Puchinger, J., Raidl, G.R.: An Evolutionary Algorithm for Column Generation in Integer Programming: An Effective Approach for 2D Bin Packing. In Yao, X., et al., eds.: Parallel Problem Solving from Nature - PPSN VIII. Volume 3242 of LNCS. Springer (2004) 642–651
11. Gilmore, P.C., Gomory, R.E.: Multistage Cutting Stock Problems of Two and More Dimensions. Operations Research **13**(1) (1965) 94–120
12. Oliveira, J.F., Ferreira, J.S.: A faster variant of the Gilmore and Gomory technique for cutting stock problems. Belgian Journal of. Operational Research, Statistics and Computer Science **34**(1) (1994) 23–38
13. Monaci, M., Toth, P.: A Set-Covering-Based Heuristic Approach for Bin-Packing Problems. INFORMS Journal on Computing **18**(1) (2006) 71–85
14. Alvarez-Valdes, R., Parajon, A., Tamarit, J.M.: A computational study of LP-based heuristic algorithms for two-dimensional guillotine cutting stock problems. OR Spectrum **24**(2) (2002) 179–192
15. Puchinger, J., Raidl, G.R.: Models and algorithms for three-stage two-dimensional bin packing . European Journal of Operational Research **183**(3) (2007) 1304–1327
16. Herz, J.C.: Recursive Computational Procedure for Two-dimensional Stock Cutting. IBM Journal of Research and Development **16**(5) (1972) 462–469
17. Christofides, N., Whitlock, C.: An algorithm for two-dimensional cutting problems. Operations Research **25** (1977) 30–44
18. Lodi, A., Martello, S., Vigo, D.: Recent advances on two-dimensional bin packing problems. Discrete Applied Mathematics **123**(13) (2002) 379 – 396
19. Hansen, P., Mladenović, N.: Variable Neighborhood Search. In Glover, F., Kochenberger, G., eds.: Handbook of Metaheuristics. Volume 57 of International Series in Operations Research & Management Science. Springer (2003) 145–184
20. Ahuja, R.K., Ergun, O., Orlin, J.B., Punnen, A.P.: A survey of very large-scale neighborhood search techniques. Discrete Applied Mathematics **123**(1-3) (2002) 75–102
21. Schrimpf, G., Schneider, J., Stamm-Wilbrandt, H., Dueck, G.: Record Breaking Optimization Results Using the Ruin and Recreate Principle. Journal of Computational Physics **159**(2) (2000) 139 – 171
22. Benjamin, J., Julstrom, B.A.: Breaking ties with secondary fitness in a genetic algorithm for the bin packing problem. In: Genetic and Evolutionary Computation Conference. (2010) 657–664
23. Lodi, A., Martello, S., Vigo, D.: Models and bounds for two-dimensional level packing problems. Journal of Combinatorial Optimization **8** (2004) 363–379
24. Berkey, J.O., Wang, P.Y.: Two-Dimensional Finite Bin-Packing Algorithms. The Journal of the Operational Research Society **38**(5) (1987) 423–429
25. Martello, S., Vigo, D.: Exact Solution of the Two-Dimensional Finite Bin Packing Problem. Management Science **44**(3) (1998) 388–399