Solving the 3-Staged 2-Dimensional Cutting Stock Problem by Dynamic Programming and Variable Neighborhood Search

Frederico Dusberger and Günther R. Raidl [1]

Institute of Computer Graphics and Algorithms
Vienna University of Technology, Vienna Austria

Abstract

We present a variable neighborhood search (VNS) for the 3-staged 2-dimensional cutting stock problem employing "ruin-and-recreate"-based very large neighborhood search in which parts of the incumbent solution are destroyed and rebuilt using construction heuristics and dynamic programming. Experimental results show that for instances where the sizes of the elements are not too small compared to the sheet size the hybridization of VNS with dynamic programming significantly outperforms a VNS relying solely on construction heuristics.

Keywords: cutting stock, variable neighborhood search, dynamic programming

1 Introduction

The two-dimensional cutting stock problem (2CS) occurs in many real-world applications such as industrial glass, paper or steel cutting, container loading, VLSI design, and various scheduling tasks [6]. In the 2CS the aim is to cut a set of small rectangular elements from larger stock sheets using as few sheets

[1] Email: {dusberger|raidl}@ads.tuwien.ac.at

as possible. 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. Furthermore, the number of *stages* during which only cuts in one direction orthogonal to the cuts in the previous stage are allowed, is often limited. For a restriction to K stages the cutting process is referred to as a K-staged cutting. In practice, often a further final stage for separating the elements from the waste is allowed.

In this work we present a variable neighborhood search (VNS) for the 3-staged 2CS which is composed of several very large neighborhood structures based on the "ruin-and-recreate" principle and dynamic programming. The general VNS framework is based on the approach presented in [3].

2 Problem Definition

In the 2CS we are given a set of n_E rectangular *element types* $E = \{1, \dots, n_E\}$ where each element type is specified by a height $h_i \in \mathbb{N}^+$, a width $w_i \in \mathbb{N}^+$ and a demand $d_i \in \mathbb{N}^+$. Furthermore, we have a (potentially unlimited) stock of identical rectangular sheets of height $H \in \mathbb{N}^+$ and width $W \in \mathbb{N}^+$. Elements are rotatable by 90°, which is reflected by adding for each type $i \in E$ a rotated type $n_E + i$ with $(h_{n_E+i}, w_{n_E+i}) = (w_i, h_i)$.

The objective is to find a *cutting pattern* P, i.e. an arrangement of the elements in E 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. 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*, the ones resulting from stage-2 cuts as *stacks* and the target rectangles specified by E as elements. A cutting pattern is specified by a cutting tree, which is detailed in Section 2.1.

We define the upper left corner of a stock sheet to have coordinates $(0, 0)$ and the lower right corner with coordinates (H, W). For each used sheet $j = 1, \dots, n$, let ρ_j be the height of the possibly remaining waste-strip at the sheet's bottom ($\rho_j = 0$ if there is no such remainder).

The objective function $c(P)$ considers not only the number of used sheets, but also the largest remaining waste strip:

$$c(P) = \min \left(n - \frac{\max_{j=1,\dots,n} \rho_j}{H} \right). \tag{1}$$

2.1 Cutting Tree

A cutting pattern is represented by its *cutting tree*, where each node represents a certain rectangle (h, w) obtained by a series of guillotine cuts. Note that a

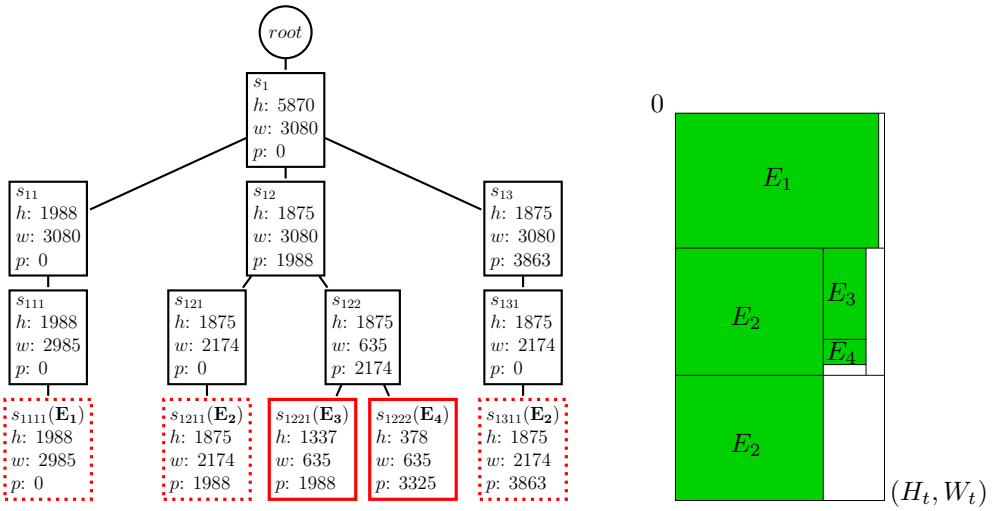

Fig. 1. A cutting tree (left) representing a single sheet, s_1, and the corresponding three-staged pattern in normal form (right). The leaf nodes represent elements of the types E_1 to E_4 obtained by the applied cutting pattern. The nodes drawn with a dotted border were introduced to keep the structure consistent.

given cutting tree does not necessarily represent a complete solution, but can also be used for partial solutions or intermediate steps.

The root node, defined to be at level -1, represents the whole cutting pattern and has one child for each used sheet. Each internal node has children corresponding to the rectangles resulting from guillotine cuts on the rectangle represented by it. One specific level k of the tree represents therefore exactly those rectangles obtained after k stages of guillotine cuts. To ensure that elements always appear as leaf nodes, additional nodes with only one child are introduced whenever an element is already finished before the 3rd stage.

Each cutting pattern P can be transformed into *normal form*, by moving waste rectangles to the bottom of stacks, the right end of strips and the bottom of sheets, respectively. Furthermore, strips and elements are ordered by nonincreasing width, stacks by nonincreasing height in the rectangle they are cut from and the sheets themselves are ordered by nondecreasing waste-strip heights ρ_j. Figure 1 shows a cutting pattern for a single sheet in normal form and the cutting tree representing it.

3 Related Work

Several state-of-the-art algorithms for the 2CS are based on a set covering formulation using variables for all possible sheet patterns, which is solved

with the help of column generation [9,2]. The first formulation of this kind has been described by Gilmore and Gomory [4] for the 2-staged 2CS. Based on this work several heuristic as well as exact approaches have been proposed in the literature, one of the crucial differences between them being how the pricing problem for the column generation is solved. Since for the 2CS this problem consists of a 2-dimensional knapsack problem with guillotine constraints a promising method to solve it is dynamic programming. Successful applications have been showed by Cintra et al. [2] and Morabito et al. [8].

As the main drawback of dynamic programming is certainly the vast number of positions where a cut on a given rectangle can potentially be applied, Herz [5] presented the concept of discretization points restricting the placement of the guillotine cuts. Scheithauer [10] further reduced the possibilities with his definition of so-called *reduced raster points*.

Besides the mentioned approaches there are, of course, also other heuristic algorithms for the 2CS. Lodi et al. [6] give a survey on common construction heuristics, such as *first-fit decreasing height* and *finite first-fit*.

4 A Dynamic Programming-Based VNS

The neighborhoods for our VNS are defined by *very large(-scale) neighborhood search* (VLNS). More precisely, they follow the "ruin-and-recreate" principle where one iteration consists of destroying randomly chosen or weak parts of an incumbent solution followed by a recreation by some optimization procedure. For this purpose we employ construction heuristics and dynamic programming.

An initial solution is constructed by running three simple construction heuristics, *3-staged First Fit Decreasing Height with rotations* (3SFFDHR), 3SFFDHR preceded by a matching step (MATCH) and *Fill Strip* (FS) and choosing the best solution obtained from them. All three heuristics are based on a finite-first fit approach, more details can be found in [3].

4.1 Neighborhood Structures and Search

All neighborhoods we consider follow the ruin-and-recreate principle and have the same basic structure. First, the ruin operator removes a fixed number (δ) or a ratio (π) of level-λ subtrees in the cutting tree, s.t. the elements in their leaf nodes become free. Besides choosing random subtrees for removal a second possibility is to order them by nonincreasing waste ratio and remove those with the largest relative waste. The removed elements are then reinserted into the cutting tree by dynamic programming.

Our dynamic programming approach uses the reduced raster points introduced by Scheithauer [10]. Let X and Y denote the raster points of the height and width, respectively and let $V_k^v(x,y)$ and $V_k^h(x,y)$ be the values of the optimal cutting pattern with at most k cutting stages applied to rectangle (x,y), where the first stage consists of vertical or horizontal cuts, respectively. In case of the unrestricted 2CS, i.e. when dropping the demand bounds on the element types, these values can be computed recursively by

$$V_k^v(x,y) = \max\left\{V_{k-1}^h(x,y), \max_{y'<y, y'\in Y}\left\{V_k^v(x,y') + V_{k-1}^h(x-x',y)\right\}\right\}, \quad (2)$$

$$V_k^h(x,y) = \max\left\{V_{k-1}^v(x,y), \max_{x'<x, x'\in X}\left\{V_k^h(x',y) + V_{k-1}^v(x,y-y')\right\}\right\}, \quad (3)$$

for all $k > 0$, $x \in X$ and $y \in Y$ and

$$V_0^v(x,y) = V_0^h(x,y) = \max\left\{0, \max_{i\in E}\left\{h_i w_i \mid h_i \le y, w_i \le x\right\}\right\}. \quad (4)$$

The restriction imposed by the element type demands leads to the problem that optimal solutions might need to be based on sub-optimal partial solutions [5]. On the other hand, for an instance consisting of a single element type the recursion will clearly yield the optimal value. Thus, the higher the average ratio of actual elements to element types, the better we can expect the results of the dynamic programming algorithm to be. To minimize the number of excess elements the algorithm considers in the recursion, we store for each computed value V the elements used to obtain this value. This allows for ensuring that the recursion does not consider combinations of two subpatterns exceeding the residual demand of one or more element types. When constructing the actual cutting pattern based on the optimal values, still an excess of some element types might be chosen, which is removed in the end.

Let $D = (d_1, \ldots, d_{n_E})$ be the vector of residual demands for the available element types. Let further R be the set of the waste rectangles at the bottom of the sheets and at the end of the strips, ordered by nonincreasing area. In detail the recreate step then works as follows:

(i) For each $r \in R$ compute $V_k^o(x_r, y_r)$, where $o \in \{h, v\}$ and $k \in \{2, 3\}$ according to the level in the cutting tree, the node representing r lies in.

(ii) If $D \neq \mathbf{0}$ continue running the recursion for rectangles of dimensions (H, W) until a resulting rectangle contains more than 60% waste.

(iii) Insert the remaining free elements using the 3SFFDHR heuristic.

k	N_k-Ruin	N_k-Recreate
1	Random; $\lambda = 3$; $\delta = 2$	Next Imp.
2-3	Random; $\lambda = 2$; $\delta = k - 1$	Next Imp.
4-5	Random; $\lambda = 1$; $\delta = k - 3$	Next Imp.
6-9	Max. waste subtrees; $\lambda = 2$; $\pi = (k - 5) \cdot 0.1$	DP
10-13	Max. waste subtrees; $\lambda = 1$; $\pi = (k - 9) \cdot 0.1$	DP
14-17	Max. waste subtrees; $\lambda = 0$; $\pi = (k - 13) \cdot 0.05$	DP
18-21	Random subtrees; $\lambda = 3$; $\pi = (k - 17) \cdot 0.1$	DP
22-25	Random subtrees; $\lambda = 0$; $\pi = (k - 21) \cdot 0.05$	DP

Table 1
Neighborhoods and their order used in the VNS

We further consider smaller neighborhoods which are searched in a next improvement manner for intensification. Due to the small number of destroyed subtrees, we apply only the 3SFFDHR heuristic in the reconstruction operator. In this work we use a fixed order of neighborhoods, which is shown in Table 1.

5 Experimental Results

Our algorithm has 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.

For the computational experiments we adapted the benchmark instances from Berkey and Wang [1] (classes 1 to 6) and Martello and Vigo [7] (classes 7 to 10) as follows: Each class consists of 5 subclasses with $|E| = 4, \ldots, 20$, each element type having a demand of 10. Each of these subclasses comprises 10 instances. We compare the basic VLNS-based VNS from [3], whose neighborhoods only use construction heuristics in the recreate operator (VNS SIMPLE) and the VNS presented in this work (VNS DP). In each experiment the VNS was given 25 major iterations over all neighborhoods and a general time limit of 1000s. Both algorithms were applied five times to each instance. For each subclass the average objective values $\overline{c(P)}$ and times $\overline{t}$ were computed. The results for classes 3,4,7,8 and 10 are given in Table 2. [2]

We performed one-sided Wilcoxon signed rank tests comparing the objective values for each instance class using a 95% confidence interval. For instance classes 3,7,8 and 10 VNS DP yields significantly better results than VNS SIMPLE, while VNS SIMPLE is significantly better only for class 4. In general,

[2] A complete listing of the results for all instance classes can be found online under `https://www.ads.tuwien.ac.at/resources/results/2cs/vns2014_simple_dp.pdf`

		VNS SIMPLE		VNS DP	
Class	$\|E\|$	$\overline{c(P)}$	$\overline{t}[s]$	$\overline{c(P)}$	$\overline{t}[s]$
	4	10.27	12.8	**10.26**	12.0
	8	16.60	94.0	**16.59**	93.4
3	12	32.30	321.7	**32.09**	319.9
	16	37.02	791.8	**36.91**	758.2
	20	49.25	1000	**49.05**	1000.0
	4	**1.44**	10.6	1.45	19.3
	8	**2.28**	67.9	2.32	82.9
4	12	**4.29**	231.0	4.33	260.4
	16	**5.10**	556.4	5.11	565.9
	20	**6.57**	946.7	6.58	989.2
	4	10.72	12.9	**10.71**	15.0
	8	19.61	97.7	**19.48**	116.1
7	12	27.68	363.2	**27.40**	419.3
	16	43.61	799.6	**43.26**	900.6
	20	52.90	1000.0	**52.36**	1000.0
	4	10.88	14.4	**10.86**	16.7
	8	18.80	110.2	**18.76**	122.2
8	12	28.94	389.3	**28.65**	443.6
	16	40.12	943.0	**40.06**	961.7
	20	50.79	1000.0	**50.48**	1000.0
	4	7.58	12.4	**7.54**	13.1
	8	16.73	93.9	**16.72**	99.4
10	12	21.55	301.6	**21.36**	333.2
	16	27.95	667.4	**27.86**	708.0
	20	33.94	928.0	**33.82**	965.5

Table 2
Experimental results. The best objective value in each row is printed in bold.

VNS DP performed better for instances where the element sizes are not very small compared to the sheet size. An obvious reason for this is that dynamic programming will more likely run out of available elements if many elements potentially fit in the larger rectangles. Remarkably, the overall runtimes of both algorithms are comparable, which is due to the generally relatively small subproblems that are solved by dynamic programming.

6 Conclusions and Future Work

We investigated a VNS for the 3-staged 2CS which relies on neighborhoods based on the "ruin-and-recreate" principle. For the reconstruction of a ruined solution we use dynamic programming based on the reduced raster points by Scheithauer. Experiments show that the proposed approach is competitive in runtime and outperforms a VNS approach relying purely on heuristics for

reconstruction.

Following the observed differences in performance w.r.t. the proportion of element sizes to the sheet size we plan to develop an adaptive VNS approach choosing suitable neighborhood structures based on their past performance or directly on features of the given problem instance.

References

[1] Berkey, J. O. and P. Y. Wang, *Two-Dimensional Finite Bin-Packing Algorithms*, The Journal of the Operations Research Society **38** (1987), pp. 423–429.

[2] Cintra, G., F. Miyazawa, Y. Wakabayashi and E. Xavier, *Algorithms for two-dimensional cutting stock and strip packing problems using dynamic programming and column generation*, European Journal of Operational Research **191** (2008), pp. 61–85.

[3] Dusberger, F. and G. R. Raidl, *A Variable Neighborhood Search Using Very Large Neighborhood Structures for the 3-Staged 2-Dimensional Cutting Stock Problem*, in: *Hybrid Metaheuristics*, 2014, pp. 85–99.

[4] Gilmore, P. C. and R. E. Gomory, *Multistage Cutting Stock Problems of Two and More Dimensions*, Operations Research **13** (1965), pp. 94–120.

[5] Herz, J. C., *Recursive Computational Procedure for Two-dimensional Stock Cutting*, IBM Journal of Research and Development **16** (1972), pp. 462–469.

[6] Lodi, A., S. Martello and D. Vigo, *Recent advances on two-dimensional bin packing problems*, Discrete Applied Mathematics **123** (2002), pp. 379–396.

[7] Martello, S. and D. Vigo, *Exact Solution of the Two-Dimensional Finite Bin Packing Problem*, Management Science **44** (1998), pp. 388–399.

[8] Morabito, R. and V. Pureza, *A heuristic approach based on dynamic programming and and/or-graph search for the constrained two-dimensional guillotine cutting problem*, Annals of Operations Research **179** (2010), pp. 297–315.

[9] Puchinger, J. and G. R. Raidl, *Models and algorithms for three-stage two-dimensional bin packing* , European Journal of Operational Research **183** (2007), pp. 1304–1327.

[10] Scheithauer, G., *Equivalence and Dominance for Problems of Optimal Packing of Rectangles*, Ricerca Operativa **83** (1997), pp. 3–34.