A Scalable Approach for the K-Staged Two-Dimensional Cutting Stock Problem with Variable Sheet Size*

Frederico Dusberger and Günther R. Raidl

Institute of Computer Graphics and Algorithms
TU Wien, Vienna, Austria
{dusberger|raidl}@ac.tuwien.ac.at

Abstract. We present a new scalable approach for the K-staged two-dimensional cutting stock problem with variable sheet size, particularly aiming to solve large-scale instances from industry. A construction heuristic exploiting the congruency of subpatterns efficiently computes sheet patterns of high quality. This heuristic is embedded in a beam-search framework to allow for a meaningful selection from the available sheet types. Computational experiments on benchmark instances show the effectiveness of our approach and demonstrate its scalability.

1 Introduction

The two-dimensional cutting stock problem occurs in many industrial applications, such as glass, paper or steel cutting, container loading, and VLSI [7]. It is particularly relevant in the manufacturing industry, where large amounts of material are processed and significant commercial benefits can be achieved by minimizing the used material.

Formally, we consider in this work the *K-staged two-dimensional cutting stock problem with variable sheet size* (K2CSV) in which we are given a set of n_E rectangular *element types* $E = \{1, \dots, n_E\}$, each $i \in E$ 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 set of n_T *stock sheet types* $T = \{1, \dots, n_T\}$, each $t \in T$ specified by a height $H_t \in \mathbb{N}^+$, a width $W_t \in \mathbb{N}^+$, an available quantity $q_t \in \mathbb{N}^+$, and a cost factor $c_t \in \mathbb{N}^+$. Both elements and sheets can be rotated by 90°. A feasible solution is a set of *cutting patterns* $\mathcal{P} = \{P_1, \dots, P_n\}$, i.e. an arrangement of the elements specified by E on the available stock sheets specified by T without overlap and using guillotine cuts up to depth K only. Each pattern P_j, $j = 1, \dots, n$, has an associated stock sheet type t_j and a quantity a_j specifying how often the pattern is to be applied, i.e. how many sheets of type t_j are cut following pattern P_j. More precisely, a cutting pattern is represented by a tree structure that is detailed in Section 1.1.

In particular, we are considering here large-scale instances from industry, where the number of different element and sheet types is moderate but the

* We thank LodeStar Technology for their support and collaboration in this project.

demands of the element types are rather high. Reasonable solutions need to be found within moderate runtimes. The objective is to find a feasible set of cutting patterns $\mathcal{P}$ minimizing the weighted number of used sheets $c(\mathcal{P})$.

$$\min\ c(\mathcal{P}) = \sum_{t \in T} c_t \sigma_t(\mathcal{P}) \tag{1}$$

where $\sigma_t(\mathcal{P})$ is the number of used sheets of type $t \in T$ in the set $\mathcal{P}$.

1.1 Cutting Tree

Each cutting pattern $P_j \in \mathcal{P}$ is represented by a (cutting) tree structure. Its leaf nodes correspond to individual elements (possibly in their rotated variants) and its internal nodes are either *horizontal* or *vertical compounds* containing at least one subpattern. Vertical compounds always only appear at odd stages (levels), starting from stage 1, and represent parts separated by horizontal cuts of the respective stage. Horizontal compounds always only appear at even stages and represent parts separated by vertical cuts. Each node thus corresponds to a rectangle of a certain size (h, w), which is in case of compound nodes the bounding box of the respectively aligned subpatterns. A pattern's root node always has a size that is not larger than the respective sheet size, i.e. $h \leq H_{t_j},\ w \leq W_{t_j}$. Recall that a_j represents the quantity of sheets cut according to pattern P_j. Similarly, compound nodes store congruent subpatterns only by one subtree and maintain an additional quantity. Compounds with only one successor are required in cases where a cut is necessary to cut off a waste rectangle, otherwise they are avoided. In this tree structure, residual (waste) rectangles are never explicitly stored, but can be derived considering a compound node's embedding in its parent compound or sheet.

Each pattern $P_j \in \mathcal{P}$ can be transformed into *normal form* having equal objective value, hence it is sufficient to consider patterns in normal form only. In normal form, subpatterns of vertical (horizontal) compounds are arranged from top to bottom (left to right), ordered by nonincreasing width (height), and aligned at their left (top) edges, i.e. in case the subpatterns have different widths (heights), remaining space appears to their right (at their bottom). Figure 1 shows a 3-staged cutting pattern and the cutting tree representing it.

2 Related Work

A classical solution approach to the K2CSV has been proposed by Gilmore and Gomory [4] who employ column generation, solving the pricing problem by dynamic programming. Although more recently, both column generation and dynamic programming still have been successfully used in approaches to the K2CSV [11, 2], the high computational effort prohibits an efficient application to large-scale instances. More reasonable approaches in terms of runtime are fast construction heuristics. In their survey on solution approaches to two-dimensional cutting stock problems, Lodi et al. [8] compare the most well-known

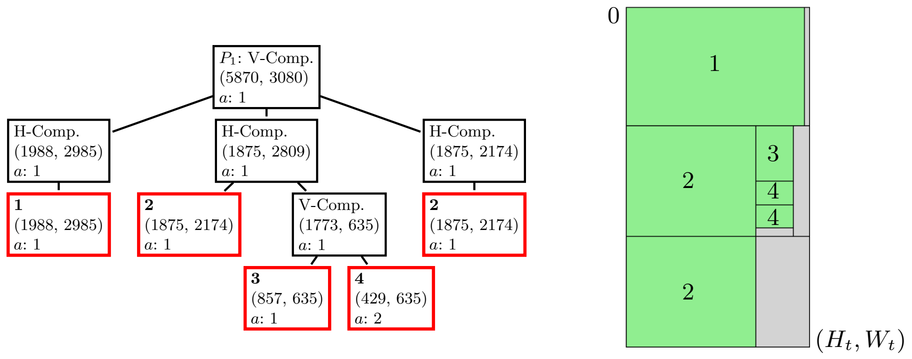
P_1: V-Comp. (5870, 3080) a: 1
H-Comp. (1988, 2985) a: 1
H-Comp. (1875, 2809) a: 1
H-Comp. (1875, 2174) a: 1
1 (1988, 2985) a: 1
2 (1875, 2174) a: 1
V-Comp. (1773, 635) a: 1
2 (1875, 2174) a: 1
3 (857, 635) a: 1
4 (429, 635) a: 2
0
1
2
3
4
4
2
(H_t, W_t)

Fig. 1: A three-staged cutting tree (left) and the corresponding cutting pattern (right). The leaves represent actual elements of types $1, \ldots, 4$ obtained after at most K stages of guillotine cuts. Note that the two elements of type 4 belong to the same vertical compound and are thus only stored once.

ones, such as First-Fit Decreasing Height or Hybrid First-Fit. Being rather simple in their nature, a major drawback of these heuristics is their inflexibility. Recently, Fleszar [3] proposed three more involved construction heuristics for the K2CSV with only a single sheet type achieving excellent results in short time. A solution is constructed element by element using a sophisticated enumeration of the possible ways of adding the current element to it.

The literature on heuristic approaches for the K2CSV specifically (i.e. when multiple sheet types are given) is rather scarce. Only recently, Hong et al. [5] embedded fast construction heuristics in a backtracking framework to address the problem of a meaningful sheet type selection. Several solutions are computed in a sheet by sheet manner and the best one is returned. Backtracking is applied to choose a different sheet type, if a choice leads to a poor solution. Another promising technique, generally applicable to problems where solutions can be computed in terms of sequences, is beam-search, which was first applied in speech recognition [9]. Beam-search is a heuristic breadth-first tree-search algorithm that only further explores a most promising restricted-size subset of nodes at each level.

3 A Congruency-Aware Construction Heuristic

In the following, we describe our heuristic for computing a pattern for a given sheet. One of the major drawbacks of conventional construction heuristics is their lack of scalability. A solution is usually constructed element by element leading to rather high runtimes for large-scale instances with high demands. To overcome this problem, we exploit congruent subpatterns during the solution construction using the following principle: The heuristic operates not on single elements but on element types. When considering a certain element type $i \in E$ for insertion into a compound c, we attempt to simultaneously insert multiple instances of i. This is realized by inserting a completely filled grid of $a_i^{\text{vert}} \times a_i^{\text{hor}}$

instances of i in c and doing the same for the compounds congruent to c. The congruent compounds can be determined by considering the quantities along the path from the current node representing c to the root of the cutting tree. Let accum_a be the accumulated quantity over these congruent compounds and let d_i^r be the *residual demand* of element type i. We can then, in total, insert

$$\min\left\{\left\lfloor\frac{a_i^{\mathrm{vert}}\cdot a_i^{\mathrm{hor}}}{\mathrm{accum}_a}\right\rfloor, d_i^r\right\} \tag{2}$$

instances of i at once. Figure 2 demonstrates this principle on a simple example.

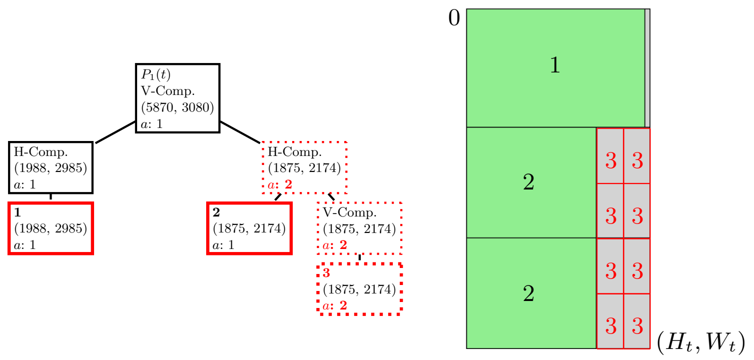

Fig. 2: Congruency-aware pattern insertion: Considering the insertion of element type 3, eight instances can be added at once by exploiting the quantities in the cutting tree.

Another drawback of conventional construction heuristics we aim to avoid is their inflexibility w.r.t. already placed elements. In particular, the sizes of the (sub-)patterns currently in the pattern tree, as well as its general structure, are usually fixed. Towards a more flexible construction heuristic, we therefore extend the concept of Fleszar [3] as follows:

For a pattern p, let $h_p^{\max}$ ($w_p^{\max}$) denote the maximum height (width) of p. The *maximum rectangle* of p is the largest rectangle to which the size of p can be extended such that it remains feasible. Let further $\tilde{h}_p$ ($\tilde{w}_p$) denote the *vertical* (*horizontal*) *slack* of p, which is the difference between p's maximum height (width) and its current height (width). The maximum rectangles and the slack values allow us to determine how much space there is left in a given compound – considering a possible resizing of it – for the insertion of a new subpattern.

A further increase of flexibility is achieved by considering not only insertions as a subpattern of a given compound, but also so-called *in-parallel* insertions. For a given compound c and a subsequence S of c's subpatterns, the $a_i^{\mathrm{vert}} \times a_i^{\mathrm{hor}}$ grid is joined in parallel to S, which allows restructuring the pattern. For further details we refer to [3].

When an element type $i \in E$ is considered for insertion, two basic questions need to be addressed: Where should the insertion be made and how many instances of i should be added, i.e. what is the best size of the grid? To answer these questions, we define the fitness of an insertion in a certain compound c as follows: Let $\tilde{h_c}$ and $\tilde{w_c}$ be the vertical and horizontal slacks of the compound after the insertion of $a_i \leq d_i^r$ instances of i, arranged in a completely filled $a_i^{\text{vert}} \times a_i^{\text{hor}}$ grid, i.e. $a_i = a_i^{\text{vert}} \cdot a_i^{\text{hor}}$. Furthermore, let η be the absolute difference between the total height (width) of the inserted grid and the height (width) of the grid's neighbor in c's normal form, given that c is a horizontal (vertical) compound. This neighbor is either the predecessor or the successor, whichever yields the smaller difference. The fitness function is then defined as

$$f(\tilde{h_c}, \tilde{w_c}, \tilde{n}) = \frac{1}{(\tilde{h_c}+1) \cdot (\tilde{w_c}+1) \cdot (\eta+1)}. \tag{3}$$

As an exception we define $f(\tilde{h_c}, \tilde{w_c}, \tilde{n}) = -1$ if the size of the grid exceeds the available space in c, i.e. if no insertion is possible. To determine the best compound c and the best size of the $a_i^{\text{vert}} \times a_i^{\text{hor}}$ grid for the insertion, we select the compound and quantities that maximize the fitness function (3), i.e.

$$\text{argmax}_{c, a_i^{\text{vert}}, a_i^{\text{hor}}}\ f(\tilde{h_c}, \tilde{w_c}, \tilde{n}) \tag{4}$$

3.1 Congruency-Aware Critical-Fit Insertion Heuristic

Based on these ideas, we propose the *congruency-aware critical-fit insertion heuristic* (CCF) extending the approach by Fleszar [3], which works as follows:

1. Order the element types in E by nonincreasing area and let further $D = \langle d_1^r, \ldots, d_{n_E}^r \rangle$ be the vector of residual demands, for all $i \in E$.
2. While $D \neq \mathbf{0}$ repeat steps 3 to 5.
3. An element type $i \in E$ *dominates* another type $j \in E$ if $w_i \geq w_j$ and $h_i \geq h_j$. Let U be the set of undominated element types considering only those types for which $d_i^r > 0$.
4. For each $i \in U$, determine the number of sheets in which i fits. Let the *critical element type* i^* be the one that fits in the least number of sheets.
5. Perform the insertion of i^* that has the highest fitness value. If no insertion of i^* is possible, a new sheet is started. Afterwards, decrease $d_{i^*}^r$ accordingly.

4 Sheet Type Selection by Beam-Search

A crucial aspect of solving the K2CSV is a meaningful selection of the sheet types on which the patterns are computed. To address this issue, we employ a beam-search strategy that generates a solution sheet by sheet. Each node in the search tree corresponds to a (partial) solution, starting with the empty solution at the root. A branch from a node reflects the decision for one of the sheet types from T, including the possibly rotated variants. At each level, all the nodes on that level are evaluated and all but the BW best ones are pruned, where BW is the chosen beam-width. Note that some nodes might also be pruned earlier, e.g. if there is no more sheet available of a certain type. This procedure continues until all residual demands are zero. Figure 3 shows an example for BW = 2.

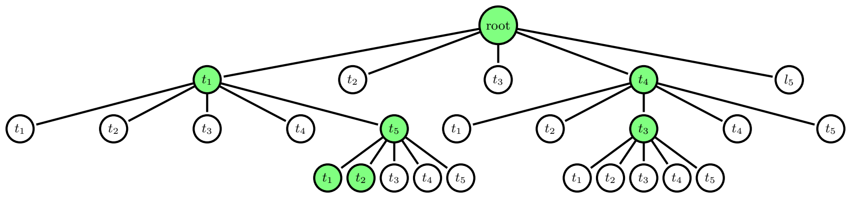

Fig. 3: Beam-search example for BW = 2. At each level, each node is evaluated by computing a pattern on the sheet type corresponding to it. The search is continued only for the two best (highlighted) nodes from that level, the remaining ones are pruned.

We compute the pattern for a single sheet using the following variant of the CCF heuristic (cf. Section 3.1): As there is only one sheet to fill, the critical element type is simply the one having the highest fitness. If there is no $i \in U$ that fits, consider all $i \in E$. Finally, if no type $i \in E$ fits, the pattern is considered finished. Exploiting congruency, the pattern resulting from the evaluation of a certain sheet type is considered to be used as often as possible. Let $\text{elems}_i(P_j)$ be the number of elements of type $i \in E$ contained in the computed pattern P_j, let t_j be the type of the sheet P_j is defined on and let d_i^r be the residual demand of i before the computation of P_j. The usable quantity for pattern P_j is then

$$\min\left\{\min_{i\in E}\left\lfloor\frac{\text{elems}_i(P_j)}{d_i^r}\right\rfloor, q_{t_j}\right\} \tag{5}$$

4.1 Node Evaluation

To evaluate the tree nodes the following criterion is applied: Let $\mathcal{P}$ be the set of patterns in the current partial solution, i.e. the patterns selected when following the path in the search tree from the root down to the current node. Furthermore, $\text{wr}(P_j)$ denotes the waste ratio of pattern P_j, i.e. the unused area on the sheet, c_{t_j} is the cost factor of the sheet type used for pattern P_j and a_j is the quantity of P_j. We prefer (partial) solutions for which the average waste ratio over $\mathcal{P}$, weighted by the respective cost factors is comparatively smaller. Formally, this ratio is defined by:

$$\frac{\sum_{P_j\in\mathcal{P}} \text{wr}(P_j)\cdot c_{t_j}\cdot a_j}{\sum_{P_j\in\mathcal{P}} c_{t_j}\cdot a_j}. \tag{6}$$

5 Computational Results

Our algorithms have been implemented in C++, compiled with GCC version 4.8.2 and executed on a single core of a 3.40 GHz Intel Core i7-3770. For all experiments, the stage limit was set to $K = 10$.

First, we investigated the effectiveness of congruency-aware pattern insertion on instances with only a single sheet type. We adapted the randomly generated benchmark instances from Berkey and Wang [1] (classes 1 to 6) and Martello

		FF		CFF		CCF				FF		CFF		CCF	
	$\lvert E\rvert$	$\overline{c(\mathcal{P})}$	$\overline{t[s]}$	$\overline{c(\mathcal{P})}$	$\overline{t[s]}$	$\overline{c(\mathcal{P})}$	$\overline{t[s]}$		$\lvert E\rvert$	$\overline{c(\mathcal{P})}$	$\overline{t[s]}$	$\overline{c(\mathcal{P})}$	$\overline{t[s]}$	$\overline{c(\mathcal{P})}$	$\overline{t[s]}$
Class 5	4	1442.3	0.2	1450	< 0.1	**1417.4**	< 0.1	Class 7	4	1091.7	0.1	1099.9	< 0.1	**1087.9**	< 0.1
	8	2285.6	0.5	2318.4	< 0.1	**2190.7**	< 0.1		8	2016	0.4	2030.3	< 0.1	**1949.3**	< 0.1
	12	4197.6	1.4	4213.4	< 0.1	**4071.9**	< 0.1		12	2870	0.9	2885	< 0.1	**2748.1**	< 0.1
	16	4983.6	2.2	5015.2	< 0.1	**4857.3**	< 0.1		16	4533.8	1.9	4541.4	< 0.1	**4401.4**	< 0.1
	20	6344.8	3.5	6366	< 0.1	**6212.8**	< 0.1		20	5442.8	2.9	5466.5	< 0.1	**5215.2**	< 0.1
Class 6	4	126.6	< 0.1	128.6	< 0.1	**124.5**	< 0.1	Class 8	4	1162.7	0.1	1167.3	< 0.1	**1136.2**	< 0.1
	8	199.3	0.1	200.7	< 0.1	**194.5**	< 0.1		8	1989.3	0.4	1988.3	< 0.1	**1887.3**	< 0.1
	12	375.5	0.2	377.4	< 0.1	**371.4**	< 0.1		12	3052.8	1.0	3063.7	< 0.1	**2949.5**	< 0.1
	16	447.2	0.3	447.5	< 0.1	**440.6**	0.1		16	4241.8	1.9	4248.4	< 0.1	**4059.8**	< 0.1
	20	582.7	0.4	583.4	< 0.1	**573.7**	0.1		20	5273.3	2.9	5297.1	< 0.1	**5178.2**	< 0.1

Table 1: Experimental results for classes 5 and 6 from [1] and classes 7 and 8 from [10]. The best objective value for each subclass is printed in bold.

			HHA		BS (BW = 150)		BS (BW = 500)	
Instance Category	$\lvert E\rvert$	$\lvert T\rvert$	$\overline{a(\mathcal{P})}$	$\overline{t[s]}$	$\overline{a(\mathcal{P})}$	$\overline{t[s]}$	$\overline{a(\mathcal{P})}$	$\overline{t[s]}$
M1	100	6	98.4	60	98.4	1.36	98.4	4.10
M2	100	6	95.6	60	95.7	1.18	96.3	3.56
M3	150	6	97.4	60	96.5	3.51	96.5	10.70

Table 2: Comparison of area utilization for the three instance categories M1 to M3

and Vigo [10] (classes 7 to 10) as follows: Each class consists of 5 subclasses with $|E| = 4, \ldots, 20$ and $d_i = 1000$, for all $i \in E$. Each of these subclasses comprises 10 instances. We compared a simple first-fit heuristic proceeding element by element (FF), a congruency-aware first-fit heuristic that always naively adds as many instances of an element type as possible at once (CFF), and the CCF heuristic. For each subclass, the average objective values $\overline{c(\mathcal{P})}$ and runtimes $\overline{t}$ were computed. In Table 1 we give the results for classes 5,6,7 and 8. As expected, for CFF we observe a significant speed-up over FF, but also an overall worse solution quality. Remarkably, the much more involved CCF heuristic has runtimes in the same order of magnitude and additionally yields significantly better solutions for each subclass. The same holds for the remaining classes.

Second, we tested our beam-search approach on the benchmark set by Hopper and Turton [6]. The set comprises 3 categories of increasing complexity. Each category consists of 5 randomly generated instances with $|T| = 6$, $2 \leq q_t \leq 4$ for all $t \in T$ and $d_i = 1$ for all $i \in E$. We compared our beam-search approach (BS) with the HHA algorithm by Hong et al. [5] as it yields – to the best of our knowledge – the so far best results on these instances in the literature and due to its relatedness to our approach. We experimented with several values for BW and chose BW = 150 and BW = 500 as the best compromises for runtime and solution quality, respectively. In Table 2 we report for each category the average percentage of the used area on the sheets $\overline{a(\mathcal{P})}$ and the average runtime $\overline{t}$ to be comparable with the results reported in [5]. Although we cannot exploit congruency, as no element type is needed more than once, our runtimes are significantly lower for all categories, compared to HHA, which always runs for $60s$. Nonetheless, our approach is competitive to HHA, achieving the same solution quality for category M1, surpassing it for category M2, and falling slightly behind for category M3. This even holds for BW = 150. Despite better runtimes, the solution quality only declines for M2, still surpassing the result from HHA.

6 Conclusions and Future Work

We presented a scalable approach for the $K2$CSV based on the exploitation of congruency during solution construction. The underlying construction heuristic scales, in principle, to arbitrarily high element type demands generating very reasonable results within a few milliseconds. In the light of multiple sheet types, beam-search has been shown to be an effective approach for a meaningful selection of the types to use, even though the underlying construction heuristic could not use its full potential.

Our congruency-aware construction heuristic poses a solid basis for the subsequent application of metaheuristics such as variable neighborhood search, where the heuristic is called very often. In future work, we thus plan to extend our approach by a respective improvement heuristic that is applied to the initially constructed solution.

References

1. Berkey, J.O., Wang, P.Y.: Two-Dimensional Finite Bin-Packing Algorithms. The Journal of the Operational Research Society 38(5), 423–429 (1987)
2. 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), 61–85 (2008)
3. Fleszar, K.: Three Insertion Heuristics and a Justification Improvement Heuristic for Two-Dimensional Bin Packing with Guillotine Cuts. Computers & Operations Research 40(1), 463–474 (2013)
4. Gilmore, P.C., Gomory, R.E.: Multistage Cutting Stock Problems of Two and More Dimensions. Operations Research 13(1), 94–120 (1965)
5. Hong, S., Zhang, D., Lau, H.C., Zeng, X., Si, Y.: A hybrid heuristic algorithm for the 2D variable-sized bin packing problem. European Journal of Operational Research 238(1), 95–103 (2014)
6. Hopper, E., Turton, B.C.H.: An Empirical Study of meta-Heuristics Applied to 2D Rectangular Bin Packing - Part I. Studia Informatica Universalis 2(1), 77–92 (2002)
7. Lodi, A., Martello, S., Monaci, M.: Two-dimensional packing problems: A survey. European Journal of Operational Research 141(2), 241–252 (2002)
8. Lodi, A., Martello, S., Vigo, D.: Recent advances on two-dimensional bin packing problems. Discrete Applied Mathematics 123(13), 379–396 (2002)
9. Lowerre, B.T.: The Harpy Speech Recognition System. Ph.D. thesis, Carnegie Mellon University, Pittsburgh, PA, USA (1976), aAI7619331
10. Martello, S., Vigo, D.: Exact Solution of the Two-Dimensional Finite Bin Packing Problem. Manage. Sci. 44(3), 388–399 (1998)
11. Pisinger, D., Sigurd, M.: The two-dimensional bin packing problem with variable bin sizes and costs. Discrete Optimization 2(2), 154–167 (2005)