FACHHOCHSCHULE TRIER
Hochschule für Technik, Wirtschaft und Gestaltung
University of Applied Sciences
Informatik

Informatik-Bericht Nr. 2008-1

Schriftenreihe Fachbereich Informatik, Fachhochschule Trier

A Local-Search Approach for Solving a Bin-Packing Problem with Secondary Objectives

Sebastian Niemann [*] Heinz Schmitz [*]

Abstract. This paper reports on a bin-packing problem that appears during the production process of wooden door frames. For this, slats are sawn out of standard-sized wood panels and the main objective is to minimize the number of used panels. However, in the case at hand two more objectives need to be considered: The ordering of slats with the same width is crucial for machine-setup costs and final-assembly costs in forthcoming stages of the production process.

After a precise problem statement we present local-search based algorithms that minimize the secondary objectives without increasing the number of panels. It turns out that a *selective neighborhood search* is an efficient but yet effective way to achieve this. Together with a standard algorithm for bin packing (FFD), our algorithms are implemented as a software module that computes optimized production schedules. Tests with real-world data show significant improvements – in comparison to a commercial optimization tool that has been used before.

Keywords: combinatorial optimization, bin packing with secondary objectives, industrial application, local search, 1-dimensional SSSCSP.

1 Introduction

In this paper we present algorithms to solve the problem of generating production schedules for a large factory that produces wooden door frames. A door frame is the fixed part of a door that holds the locking plate and covers the wall, see Fig. 1. Such a frame consists of three slats of equal width that in turn depends on the thickness of the wall. In each of the three stages of the main production process a certain optimization goal has to be considered.

Stage 1. Here slats from different customer orders are grouped together according to the type of wood. For each such group slats are sawn out of standard-sized wood panels. This is a one-dimensional cutting process. Since it is not always possible to allocate the slats in a way such that each panel is completely utilized there will typically be some cut-off left on each panel which then has to be chipped by the saw. Fig. 2 shows a stack of panels before and after the cutting process in the first stage. The natural first objective is to minimize the number of panels.

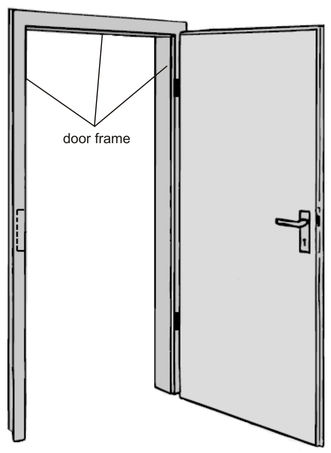

Figure 1: Schematic illustration of a door.

Stage 2. In the second stage the slats are serialized and handled one by one to finish them, see again Fig. 2. They are reamed to size and a flute is implemented to add a cover. The machine that carries out this step has to be changed over every time the width of consecutive slats changes. In such a case fixed and variable setup times occur, the latter depending on the quantity of the width difference. As a second objective we identify the minimization of setup times for this second stage.

Stage 3. Here the door frames are assembled. As depicted in Fig. 1 three slats of equal width are required for each frame. To do so, it is favorable to have multiples of three slats of equal width close together. If this is not the case an employee has to sort the slats manually which is very time-consuming. The third objective taken into account is the minimization of (estimated) sorting times in the final assembly.

A schedule is completely determined by the distribution of slats to panels in the first stage. The ordering of slats is the same during the whole process (excpect for inversion) because no sorting is possible while moving stacks of sawn panels. So all values of the objective functions are already known when the distribution of slats to panels is assigned. While the second and third objectives are similar (but not equal), they are competing with the first one.

[*]Both authors are with the Department of Computer Science, Fachhochschule Trier, Schneidershof, D–54293 Trier, Germany (email: niemanns@fh-trier.de, schmitz@informatik.fh-trier.de)

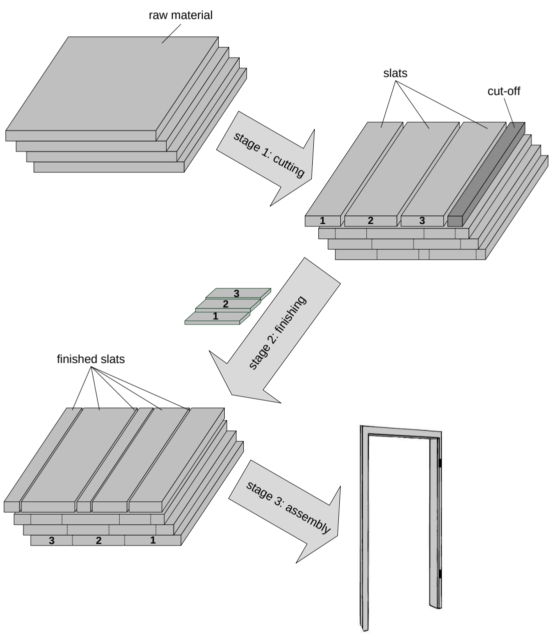

Figure 2: Three-stage production process of door frames.

As is typical for real-world problems some additional technical characteristics have to be considered. For example each panel has to be straightened before the slats are sawn which reduces the panel width by an edge crop width. Furthermore, even when sawing single slats some cut-off is produced (crop width). These widths depend on the type of wood so they are part of the input data for each instance.

There are two reasons why the first objective dominates the others: On one hand various machine characteristics make the production process much more complicated when too much panels are used (hence a lot of cut-off is produced). On the other hand raw panels are expensive compared to the machine and assembly costs in the forthcoming stages. Consequentially, we can characterize the problem as being of type $Lex(z_1, F_l(z_2, z_3))$ where z_i is the objective of stage i and F_l is a weighted linear function (for notations see, e.g., [9, p.110]).

After a precise problem statement, we propose a combination of FFD and a subsequent local-search variant to solve it. It turns out that our approach is a conceptually easy, but yet efficient and effective way to outperform the formerly used single-objective commercial optimizer (and to make manual post-optimization obsolete). As a result, this commercial tool has been replaced by our optimization module.

Related Work. While single-criteria cutting stock problems were among the first to be studied in operations research, more and more multicriteria variants have been considered since the mid-eighties. Several additional objectives have been taken into account, e.g., the minimization of the total number of different cutting patterns [6, 10, 5] and the heterogeneousness of the used patterns [4]. Furthermore, secondary objectives for bin packing problems have been studied , e.g., balancing the load of each bin [8]. Besides one-dimensional problems also two-dimensional variants of cutting and bin-packing problems are discussed in the literature, see e.g. [7].

The problem at hand can be classified by considering the improved typology of cutting and packing problems [11]. Here, problems are mainly characterized by five criteria in three steps (basic, intermediate and refined problem types). Considering only the main objective of minimizing material usage our problem can be characterized as a CSP (basic type) with several identical large objects (intermediate type) and one dimension (refined type). This leads to a 1-dimensional Single Stock Size Cutting Stock Problem (1-dimensional SSSCSP). However, problems with multiple objectives are handled as *variants* in the typology.

2 Problem Statement

We formally define the present optimization problem as follows. An instance $x = (m, S, P)$ consists of a list of m slats $S = (w_1, \dots, w_m)$ where $w_i \in \mathbb{N}$ defines the width of slat i, and a list of parameters P. The parameter tupel $P = (pw, cw, ecw, maxparts, as, fst, st, \alpha_2, \alpha_3)$ has the following meaning:

pw	**p**anel **w**idth
cw	**c**rop **w**idth
ecw	**e**dge **c**rop **w**idth
$maxparts$	**max**imum number of **parts** per panel
as	**a**djustment **s**peed (second stage)
fst	**f**ix **s**etup **t**ime (second stage)
st	**s**orting **t**ime (third stage)
α_2	weight for second objective
α_3	weight for third objective

A solution for x is a cutting plan $p = (l_1, \dots, l_k)$ with k layouts where each layout $l_i = (w'_{i1}, \dots, w'_{iq_i})$ with widths $w'_{ij} \in \mathbb{N}$ defines a cutting pattern.

For an easy notation of constraints and objectives let

$$\begin{aligned} & (w'_{11}, w'_{12}, \dots, w'_{1q_1}, \dots, w'_{k1}, \dots, w'_{kq_k}) \\ = & (b_1, \ b_2, \dots \qquad\qquad\qquad\qquad , b_m) \\ = & I(p) \end{aligned}$$

be the list of slat widths in processing sequence. Feasible solutions have to satisfy several constraints. First of all the panel width given by the parameters must not be exceeded. Of course we have to take the (edge) crop widths into account when calculating the used panel width, so we have

$$\forall 1 \le i \le k: \left(\sum_{j=1}^{q_i} w'_{ij} + (q_i - 1) * cw + 2 * ecw \right) \le pw \quad (1)$$

where q_i is the number of slats on panel i. As a second constraint this number of slats must be bounded above by the parameter *maxparts* for each layout, i.e.

$$\forall 1 \le i \le k : q_i \le maxparts. \quad (2)$$

As no overproduction is allowed, each slat must be scheduled exactly once, that is

$$\forall 1 \le i \le m: |\{j \mid w_j = w_i\}| = |\{j \mid b_j = w_i\}| \quad (3)$$

$$\sum_{i=1}^{k} q_k = m. \quad (4)$$

A production plan is *valid* (or executable) iff (1) - (4) hold.

As a first objective we consider the minimization of the number of panels (or layouts) used in p, so we have

$$z_1(p) = k.$$

The second objective measures the sequence dependent setup times in the second production stage. As described in Sec. 1 there are fixed and variable times. For the variable proportion we sum up the consecutive absolute width differences and divide them by the adjustment speed of the relevant machine, i.e.

$$z_{21}(p) = \left(\sum_{i=1}^{m-1} |b_{i+1} - b_i| \right) / as.$$

The fixed setup time is determined by counting the number of width changes and is therefore defined by

$$z_{22}(p) = \left(\sum_{i=1}^{m-1} d(b_{i+1}, b_i) \right) * fst$$

whereas

$$d(x,y) = \begin{cases} 1 & \text{if } x \ne y \\ 0 & \text{else.} \end{cases}$$

So the overall setup time in the second stage is

$$z_2(p) = z_{21}(p) + z_{22}(p).$$

To calculate the third objective we count non-grouped slats. Let $grp(p)$ be the number of disjoint terns of equal width in p, then

$$z_3(p) = (m - 3 * grp(p)) * st$$

is the sorting time during final assembly.

Now the task is to find among all valid production plans p with minimal $z_1(p)$ a plan where also $F_l(z_2, z_3) = \alpha_2 * z_2 + \alpha_3 * z_3$ is minimal. It is easy to see that calculating an optimal solution for the first objective is equivalent to finding an optimal solution for the classical bin-packing problem where m objects with sizes a_i (slat widths w_i) and a maximum size B (here pw) for each bin are given. The **NP**-hardness for bin packing carries over to the present multi-objective variant ([9, p.113]).

3 Algorithm Design

The different priorities of our objectives motivate an approach with subsequent optimization steps. In the first step we optimize z_1 by adapting a standard bin-packing approximation algorithm. In the second and third step we try to minimize $F_l(z_2, z_3)$ using two different strategies that maintain the number of used panels.

3.1 Step One: Panel Optimization

We ignore the secondary objectives and calculate a solution that uses a small number of panels. As this subproblem is equivalent to the bin-packing problem we are able to choose from a variety of exact, heuristic and approximation algorithms, e.g. [2], [1]. Algorithm 1 shows an implementation of **F**irst **F**it **D**ecreasing (FFD) as one such possibility. Slats are sorted in descending order and then allocated one after another on the first panel with sufficient space left.

Algorithm 1: `FFD`(x)

```
Input  : instance $x = (m, S, P)$
Output : solution $p = (l_1, l_2, \dots, l_k)$
begin
    sort $w_i$ in descending order resulting in $(\tilde{w}_1, \tilde{w}_2, \dots, \tilde{w}_m)$;
    $k := 0$;
    $p := ()$;
    for $i := 1$ to $m$ do
        if $\tilde{w}_i$ cannot be placed on any panel then
            $k := k + 1$;
            $l_k := ()$;
            $p := (l_1, l_2, \dots, l_{k-1}, l_k)$;
        $j :=$ first index of a panel $\tilde{w}_i$ can be placed on;
        $l_j := (w'_{j1}, w'_{j2}, \dots, w'_{jq_j}, \tilde{w}_i)$;
    return $p$
end
```

When searching a panel for $\tilde{w}_i$ we check the constraints described in Section 2, i.e., only panels j with sufficient space (1) and $q_j < maxparts$ (2) are selected. Since each slat is considered exactly once in the for-loop also (3) and (4) hold. So the plan generated by `FFD` is always valid. It is known that this $O(n \log n)$-algorithm guarantees a performance ratio of at most 22%, more precisely, it holds for all x that $\texttt{FFD}(x) \leq 11/9 * \texttt{OPT}(x) + 6/9$, see [3]. Experimental results with our test data show that the performance ratio is usually a lot better, even the optimum is achieved quiet often. We take this to justify the decision to implement FFD for the present problem.

3.2 Step Two: Single-Slat Optimization

The idea to improve the additional objective $F_l(z_2, z_3)$ is to determine slats with the highest potential for improvement. However, there are different possibilities to determine such positions. Several tests indicate that a greedy approach works out best that chooses the slat adding the highest cost fraction to $F_l(z_2, z_3)$. Algorithm getImpPos calculates such a position. The function $inGroup(p, i)$ returns 1 iff slat i is located in a group counted by $grp(p)$. Moreover, only unmarked positions are returned (the following local-search algorithm Alg. 3 marks positions that could not be further improved).

Algorithm 2: getImpPos(x, p)

```
Input  : instance $x = (m, S, P)$, solution
         $p = (l_1, l_2, \ldots, l_k)$
Output : position $pos$ of a slat that mostly
         contributes to $F_l(z_2, z_3)$
begin
    $max := 0$;
    $pos := -1$;
    for $i$:=2 to $m - 1$ do
        $z_{21} := (|b_{i-1} - b_i| + |b_i - b_{i+1}|) / as$;
        $z_{22} := (d(b_{i-1}, b_i) + d(b_i, b_{i+1})) * fst$;
        $z_2 := z_{21} + z_{22}$;
        $z_3 := (1 - inGroup(p, i)) * st$;
        if $F_l(z_2, z_3) > max \land i$ unmarked then
            $max := F_l(z_2, z_3)$;
            $pos := i$;
    return $pos$
end
```

Next we try to reduce the costs of a plan by exploring the neighbors of the slats determined by getImpPos. When looking at test data it turned out that the combined use of two neighborhood functions is a good idea also in this case. Let p such that $I(p) = (b_1, \ldots, b_i, \ldots, b_j, \ldots, b_m)$. The first neighborhood is the exchange neighborhood

$$\mathcal{N}_1(p, i) = \{exchg(p, i, j) \mid 1 \leq j \leq m\}$$

where $exchg(p, i, j) = p'$ such that $I(p') = (b_1, \ldots, b_j, \ldots, b_i, \ldots, b_m)$. For a second neighborhood we move a slat from position i to position j, i.e., let $move(p, i, j) = p'$ such that $I(p') = (b_1, \ldots, b_{i-1}, b_{i+1}, \ldots, b_j, b_i, b_{j+1}, \ldots, b_m)$, and

$$\mathcal{N}_2(p, i) = \{move(p, i, j) \mid 1 \leq j \leq m\}.$$

The following algorithm intensifies the search in the direction of the subsequently calculated positions getImpPos in a way we call *selective neighborhood search* (SNS). It exhaustively evaluates the neighborhoods $\mathcal{N}_1(p, pos)$ and $\mathcal{N}_2(p, pos)$ for $pos = \texttt{getImpPos}(x, p)$ with the first-improvement strategy until no more improvements are possible. We mark positions to avoid selecting them twice.

Algorithm 3: SNS:SSO(x, p)

```
Input  : instance $x = (m, S, P)$, solution
         $p = (l_1, l_2, \ldots, l_k)$
Output : improved solution
         $p = (l'_1, l'_2, \ldots, l'_k)$
begin
    repeat
        $pos$ :=getImpPos$(x, p)$;
        foreach $p' \in \mathcal{N}_1(p, pos) \cup \mathcal{N}_2(p, pos)$ do
            if $F_l(z_2(p'), z_3(p')) <$
            $F_l(z_2(p), z_3(p)) \land z_1(p') \leq z_1(p)$
            then
                $p := p'$;
                break;
        if $\neg$ improvement found then
            mark $pos$;
        else
            remove all marks;
    until all positions are marked ;
    return $p$
end
```

Call p locally optimal w.r.t $F_l(z_2(\cdot), z_3(\cdot))$ and a neighborhood $\mathcal{N}$ iff $F_l(z_2(p), z_3(p)) \leq F_l(z_2(p'), z_3(p'))$ for all $p' \in \mathcal{N}(p, i)$ and $1 \leq i \leq m$. So the result of Algorithm SNS:SSO is locally optimal with respect to $\mathcal{N}_1 \cup \mathcal{N}_2$. Note that by the choice of *pos* we look at such neighborhoods first, where we expect significant improvements. This accelerates the search towards the local optimum and makes it a very efficient procedure. Nonetheless, further improvements may be possible since $\mathcal{N}_1$ and $\mathcal{N}_2$ only consider moves and swaps of *single* slats.

Since in Algorithm 3 the neighborhood search is aborted as soon as an improvement is found, in an efficient implementation neighbors with the most room for improvement should be evaluated first. This can, for example, be done by calculating prefered swap and move positions for slats at position *pos* and evaluating the corresponding neighbors first.

3.3 Step Three: Slat-Group Optimization

To further improve $F_l(z_2, z_3)$ we now consider groups of equal-sized slats. Let p such that $I(p) = (b_1, \ldots, b_i, \ldots, b_{i+r}, \ldots, b_j, \ldots, b_m)$ with $b_{i-1} \neq b_i$, $b_i = b_{i+d}$ for all $1 \leq d \leq r$ and $b_{i+r} \neq b_{i+r+1}$. Then we define

$moveG(p, i, j) = p'$ such that $I(p') = (b_1, \ldots, b_{i-1}, b_{i+r+1}, \ldots, b_j, b_i, \ldots, b_{i+r}, \ldots, b_m)$ and

$$\mathcal{N}_3(p, i) = \{moveG(p, i, j) \mid 1 \leq j \leq m\}.$$

The next algorithm moves groups of slats by evaluating $\mathcal{N}_3$ for every group in p. An auxiliary function `getGroupPos`(x, p, pos) returns the starting index of the first group of equal-sized slats that is greater than pos, and -1 if no such index exists.

Algorithm 4: `SNS:GSO`(x, p)

```
Input   : instance $x = (m, S, P)$, solution
          $p = (l_1, l_2, \ldots, l_k)$
Output  : improved solution
          $p = (l'_1, l'_2, \ldots, l'_k)$
begin
    $pos :=$ getGroupPos$(x, p, 0)$;
    while $pos \neq -1$ do
        foreach $p' \in \mathcal{N}_3(p, pos)$ do
            if $F_l(z_2(p'), z_3(p')) <$
            $F_l(z_2(p), z_3(p)) \wedge z_1(p') \leq z_1(p)$
            then
                $p := p'$;
                $pos := 0$;
                break;
        $pos :=$ getGroupPos$(x, p, pos)$;
    return $p$
end
```

Again, this algorithm does not terminate as long as improvements are possible. As a consequence, the resulting plan p is locally optimal w.r.t. $\mathcal{N}_3$. This is final result of the overall computation.

4 Experimental Results

In order to assess our approach, also for large-scaled real-world instances, we compare the results produced by `FFD/SNS` with plans previously put in practice by the production-line manager. Except for a few special cases, these plans were calculated by a commercial optimizer (CO) specilized on the primary objective z_1. However, this software could not cope with the secondary objectives that are specific for the present problem.

Table 1 compares results for some typical real-world instances. To calculate the criteria values an adjustment speed of $as = 20$, a fix setup time of $fst = 10$ and a sorting time of $st = 5$ have been used. Furthermore, realistic weights of $\alpha_2 = 0.7$ and $\alpha_3 = 0.3$ were assumed.

It turns out that the plans generated by our `FFD/SNS`-combination yield better objective values in almost all cases. Typically, the number of used panels remains unchanged while the values for the secondary objectives are improved by up to 50% in comparison to the CO. In few cases, we even obtained a smaller number of panels. It should be mentioned that all computations were carried out in a few seconds on a standard personal computer.

ID	Criterion	CO	`FFD/SNS`	Imp.
67888 $m = 86$	z_1	23	23	0%
	z_2	287	196	32%
	z_3	115	55	52%
	$F_l(z_2, z_3)$	247	154	38%
67896 $m = 122$	z_1	37	37	0%
	z_2	1086	652	40%
	z_3	235	190	19%
	$F_l(z_2, z_3)$	831	513	38%
67932 $m = 252$	z_1	67	66	1%
	z_2	1732	1342	23%
	z_3	225	285	-27%
	$F_l(z_2, z_3)$	1280	1025	20%
67924 $m = 265$	z_1	44	44	0%
	z_2	1184	918	22%
	z_3	290	155	47%
	$F_l(z_2, z_3)$	916	689	25%

Table 1: Comparison of real-world instances. The criteria values could be improved in almost all cases.

To emphasize the impact of the `SNS`-algorithm Fig. 3 shows the objective value for $F_l(z_2, z_3)$ before and after the selective neighborhood search for a larger set of real-world instances. It can be observed that the value for F_l was improved in all considered instances, often up to 50%.

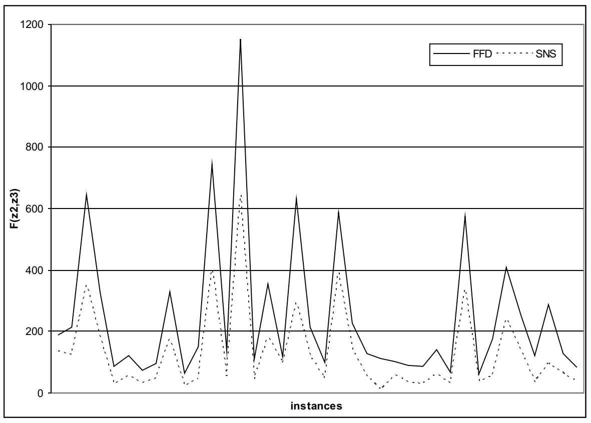

Figure 3: Comparison of criteria values before and after optimization with `SNS`.

5 Conclusions

We propose a combination of FFD and a subsequent local-search variant to solve a real-world bin-packing problem with secondary objectives that arises during the production process of wooden door frames. Not surprisingly, a commercial optimizer specialized only on the primary objective could not cope with the problem-specific secondary objectives. It turns out that our approach is a conceptually easy, but yet efficient and effective way to outperform such an optimizer (and to make manual post-optimization obsolete). As a result, the commercial tool has been replaced by our optimization module.

Future Work. It would be interesting to investigate possible performance improvements of our approach by using additional neighborhood functions. Instead of moving single slats only (or single groups of equal-sized slats) one could try to implement a heap to swap out several slats simultaneously and then search for better positions regarding all slats in the heap. Another extension is motivated by the second production line in the factory. Because of logistic constraints for this line only specific cutting patterns (layouts) may be scheduled. Since this problem is fundamentally different from the one described in this paper we are currently working on an adequate algorithm to generate plans for this production line.

References

[1] Coffman, M. Garey, and D. Johnson. Approximation algorithms for bin-packing: An updated survey. *Algorithm Design for Computer Systems Design*, pages 49–106, 1984.

[2] Coffman, M. Garey, and D. Johnson. Bin packing appoximation algorithms: Combinatorial analysis. *Handbook of Combinatorial Optimization*, 1998.

[3] G. Dosa. The tight bound of first fit decreasing bin-packing algorithm is $\mathrm{FFD(I)} \leq (11/9) * \mathrm{OPT(I)} + 6/9$. *Lecture Notes in Computer Science*, 4614:1–11, 2007.

[4] M. J. Geiger. Bin packing under multiple objectives: a heuristic approximation approach. *The Fourth International Conference on Evolutionary Multi-Criterion Optimization: Late Breaking Papers*, pages 53–56, March 2007.

[5] C. Goulimis. Optimal solutions for the cutting stock problem. *European Journal of Operational Research*, 44:197–208, 1990.

[6] A. W. J. Kolen and F. C. R. Spieksma. Solving a bi-criterion cutting stock problem with open-ended demand: a case study. *The Journal of the Operational Research Society*, 51(11):1238–1247, November 2000.

[7] C. León, G. Miranda, C. Rodíguez, and C. Segura. 2d cutting stock problem: A new parallel algorithm and bounds. *Lecture Notes in Computer Science*, 4641/2007:795–804, 2007.

[8] D.S. Liu, K.C. Tan, C.K. Goh, and W.K. Ho. On solving multiobjective bin packing problems using particle swarm optimization. *Evolutionary Computation CEC IEEE Congress*, pages 2095–2102, 2006.

[9] V. T'kindt and J.-C. Billaut. *Multicriteria Scheduling*. Springer, second edition, 2006.

[10] S. Umetani, M. Yagiura, and T. Ibaraki. One-dimensional cutting stock problem to minimize the number of different patterns. *European Journal of Operational Research*, 146:388–402, 2003.

[11] G. Wäscher. An improved typology of cutting and packing problems. *Working paper 24, University Magdeburg*, 2006.