A Heuristic for a Rich and Real Two-dimensional Woodboard Cutting Problem

Claudio Arbib[1], Fabrizio Marinelli[2], Andrea Pizzuti[2] and Roberto Rosetti[2]
[1]*Dipartimento di Scienze/Ingegneria dell'Informazione e Matematica, Università degli Studi dell'Aquila, Via Vetoio, Coppito, I-67010 L'Aquila, Italia*
[2]*D.I.I., Università Politecnica delle Marche, Via Brecce Bianche, I-60131 Ancona, Italia*

Keywords: Cutting Problem, Heuristics, Manufacturing.

Abstract: Cutting operations in manufacturing are characterized by practical requirements and utility criteria that usually increase the complexity of formulations or, even worse, are difficult to be modeled in terms of mathematical programming. However, disregarding or just simplifying those requirements often leads to solutions considered not attractive or even useless by the manufacturer. In this paper we consider a rich two-dimensional cutting stock problem that covers the whole specification of a family of wood cutting machines produced by a worldwide leader in industrial machinery manufacturing. A sequential value correction heuristic is implemented to minimize the employed stock area while reducing additional objective functions.

1 INTRODUCTION

The CUTTING STOCK PROBLEM (CSP) is a well-known model of real-world optimization problems arising in manufacturing. This difficult problem has a very simple statement: *find an efficient way of cutting small objects from large ones*.

A wide literature investigates the basic CSP and its variants, see (Wäscher et al., 2007) for a comprehensive survey. Indeed, variants exist to fill the gap between theory and practice, and a variant is regarded of interest for the academia as far as it requires a non-standard model and/or a novel methodological approach.

Unfortunately, it is often hard to encapsulate reality into a single elegant mathematical formulation. Moreover, it is generally recognized that the success of an optimization method depends on the concurrency of transversal skills: one is indeed mathematical modeling, but it is not the only one. A fundamental role is for instance played by the bi-directional interface that must be developed to feed the model with data and to present the solutions to decision-makers: a development that normally requires more IT than math skills.

On the other hand, the more a model is close to operational decision, the more attention is to be devoted to process details. Disregarding such details (which is often done in research papers in order to capture the mathematical essence of the problem) would simply cause the software product not to work, and the management not to buy it. For example, edge trimming is generally neglected in CSP models; but a cutting pattern that does not take it into account, may not be realizable in practice.

While a specification on edge trimming can easily be dealt with after prototype development, this is not the case of other apparently irrelevant issues. For example, blade thickness expressed in fractions of millimeters can increase the precision required of the packing algorithm, with important effects on its efficiency if, as usual, it has pseudo-polynomial time complexity.
A twofold challenge has therefore to be addressed:

- *On software design*. Technical issues — be they expressed as constraints or utility criteria — have both a local impact on the feasibility of cutting patterns (number of stages, item rotation etc.) and a global one on the cutting plan as a whole (number of patterns, cut sequence etc.). Pattern generation algorithms can differ a lot from each other in order to take care of the former issues, and the latter can affect the whole software architecture.
- *On decision making.* A hierarchical problem decomposition based on the various utility criteria (e.g. first minimize trim-loss, then setups, then open stacks) can output low-quality or even infea-

Arbib, C., Marinelli, F., Pizzuti, A. and Rosetti, R.
A Heuristic for a Rich and Real Two-dimensional Woodboard Cutting Problem.
DOI: 10.5220/0006534500310037
In *Proceedings of the 7th International Conference on Operations Research and Enterprise Systems (ICORES 2018)*, pages 31-37
ISBN: 978-989-758-285-1

sible solutions. On the other hand, combining different objectives into a single goal may produce solutions not appreciated by all the involved decision makers.

We here propose to address this challenge by a single architectural choice in which patterns are generated via *Sequential Value Correction* heuristics and then used to provide a collection of *non-dominated solutions*.

Problems with a level of detail that covers industrial specification as a whole and puts the solution method in a condition to operate in the plant, are called *rich*. In this paper we consider a rich Multiple-stock-size two-Dimensional CSP (M2D-CSP) compliant with the specification of a family of wood cutting machines produced by *SCM Group* (SCM, 2017).

The M2D-CSP is a variant of the CSP that can be presented as follows. Let I be a set of n distinct rectangles (part types). Each $i \in I$ has length l_i and height h_i, and must be cut in d_i copies from a stock set J of m different (and larger) rectangles. Each $j \in J$ has length L_j and height H_j, and is available in b_j copies. The cut should be done so as to minimize used stock area; other objectives — such as reduction of employed stocks, setup minimization etc. — can however be selectively or entirely considered.
Our problem has these main features:

- Cuts proceed parallel to stock item from edge to edge (*guillotine cut*).
- Items (either produced or in stock) can be orthogonally rotated.
- 2 and (simplified) 3-stage cuts are possible: in a first stage, a configuration $\mathbf{c} = \langle$stock type, first cut orientation (horizontal or vertical)$\rangle$ is selected, and multiple cuts are done orthogonally to orientation; then the stock item is rotated by 90° and the stage is repeated.
- Precut is possibly allowed as a pre-processing step.

Part types admit overproduction if explicitly indicated, but demand is in general to be fulfilled exactly. Indeed, relaxing demand constraints by overproduction can yield solutions with very small trim-loss; but a substantial number of excess items are cut, increasing too much the necessary amount of stock items and consequently production costs.

2 LITERATURE REVIEW

Real cutting processes often deal with heterogeneous stock types. The relevant M-CSP is handled in various ways. In (Belov and Scheithauer, 2002) an exact cutting plane approach is proposed for one-dimensional CSP with multiple stock lengths. The same problem is solved heuristically in (Poldi and Arenales, 2009). In (Yanasse et al., 1991), the two-dimensional CSP in the wood industry is addressed and a heuristic algorithm is proposed for the multiple stock sizes case.

Trim-loss minimization is not the only criterion considered in an industrial cutting process: indeed, setup costs and the number of stacks maintained open throughout the process can affect process quality as well. Examples of CSP seeking both setup minimization and open stack limitation can be found in (Belov and Scheithauer, 2007), where a sequential heuristic for the one-dimensional case is proposed, and in (Nonås and Thorstenson, 2000), where the problem is formulated as a non-linear concave minimization problem and solved through global or local procedures. Metaheuristics and an exact approach that separately address the CSP with setup minimization are reported respectively in (Umetani et al., 2003) and (Aloisio et al., 2011); moreover CSP with open stack limitation is considered in (Arbib et al., 2016) and (Yanasse and Lamosa, 2007).

Although industrial applications require consideration of several practical issues, these are taken into account by very few papers; and in those cases, very few additional issues are regarded simultaneously. Here is a list of exceptions (to the best of our knowledge):

- (Malaguti et al., 2014) describes a truncated Branch-and-Price algorithm to solve an M2D guillotine CSP in woodboard cutting industry, where orthogonal rotation of items and boards is allowed, and stacked boards can be cut concurrently. The objective is to minimize a weighted combination of stock usage, number of cutting cycles and of 3-stage cuts.
- (Varela et al., 2008) presents a 1D-CSP arising in a plastic manufacturing facility, with five objective functions hierarchically ranked, some technical parameters related to the employed cutting machines and order priorities. A GRASP algorithm with a call to a sequential heuristic procedure is implemented.
- (Chu and Antonio, 1999) proposes a mathematical formulation for a rich 1D-CSP in the metal industry. The various technical constraints taken into account significantly affect the total cost and involve trim-loss, material reusability and cut time. Approximation algorithms based on dynamic programming are devised.

In conclusion, it is worth quoting sequential heuristics. Instead of a rigorous (but slow) solution of the pricing problem with exact dual prices, algorithms of this type use pseudo-prices to sequentially (and quickly) compose each single patterns, and in this way can achieve good solutions. Various implementations have been proposed for 1D bin packing and CSP (Belov and Scheithauer, 2007), (Varela et al., 2008), and 2D bin-packing and CSP (Cui et al., 2015), (Chen et al., 2016).

3 A SEQUENTIAL VALUE CORRECTION HEURISTIC

The basic idea behind the sequential value correction scheme consists in assigning a value (the pseudo-price) to each part type and generate cutting pattern sequentially, each one with the parts not yet allocated that maximize the total value of the pattern. After a solution has been computed, pseudo-prices are conveniently updated and the process reiterated in order to fill a pool of non dominated solutions. Dominance relationship are based on the following minimization criteria: total used stock area, total trim-loss, number of used stocks, number of distinct setups, and total number of precuts (Notice that used area, trim-loss and number of used stocks measure distinct performance due the overproduction features and the presence of stocks of different sizes).

The whole procedure is composed by three main steps, each one optimizing a distinct performance indicator by resorting to Algorithm 1, which is a sequential value correction heuristic (SVC).
The algorithm can be summarized as following:

- Step 1: SVC solves the M2D-CSP with non-batched demand, seeking used area minimization.
- Step 2: The second step tries to reduce the number of setups by demand rounding. In particular, the demand of each part type is aggregated by increasing batch dimension (parameter *size*) and SVC is called to find solutions with reduced number of setups. Starting from $size = 2$, the procedure is iterated for increasing values of *size*, chosen as the largest integer in $[size+1, \max_{i \in I} d_i]$ that minimizes the area associated to the residual demand $\mathbf{r}'$. Step 2 terminates when SVC does not find any new non-dominated solution.
- Step 3: in this phase SVC searches for solutions with reduced number of precuts. A suitable opportunity threshold δ is used to trigger decreasing chances of employing a precut. The procedure iterates by updating δ until new generated solutions have no precut.

Algorithm 1: Sequential value correction heuristic.

```
1: procedure $SVC(\mathbf{d}, size, \delta)$
2:     $\lambda_i \leftarrow S_i$
3:     $cuts \leftarrow 0$
4:     for $i \leftarrow 1 \dots N$ do
5:         $g \leftarrow random(1, \rho)$
6:         $\mathbf{r} \leftarrow \mathbf{d}$
7:         $s = \emptyset$
8:         repeat
9:             $\mathbf{d}' \leftarrow \lfloor \frac{\mathbf{r}}{size} \rfloor$
10:            $\mathbf{r}' \leftarrow \mathbf{r} \bmod size$
11:            while $\mathbf{d}' > 0$ do
12:                $P = \emptyset$
13:                for $\mathbf{c} \in C$ do
14:                    $p[\mathbf{c}] \leftarrow getPattern(\mathbf{c}, \lambda, \delta, \mathbf{d}')$
15:                    $P = P \cup \{p[\mathbf{c}]\}$
16:                end for
17:                $(a, cuts) \leftarrow getBestPattern(P)$
18:                $fillingPattern(a, cuts)$
19:                $s \leftarrow a$
20:                $h \leftarrow random(1/g, g)$
21:                $\lambda \leftarrow updatePrices(a, \lambda, h)$
22:            end while
23:            $\mathbf{r} \leftarrow \mathbf{r}'$
24:            $size = 1$
25:        until $\mathbf{r} > 0$
26:        $patternSeq(s, cuts)$
27:        if $notDominated(s)$ then
28:            $Sol = Sol \cup \{s\}$
29:        end if
30:    end for
31: end procedure
```

3.1 SVC Heuristic

The SVC heuristic is the core of the algorithm and is implemented to build the CSP solutions.

Let I' be the set formed by the items of I and their rotated counterparts. In its general framework, SVC starts by defining a price λ_i for each part type $i \in I'$. Each price is initialized at the item area S_i. Part type demand is batched by *size*, that is, stock items are stacked and each pattern is activated at multiples of *size*. A partial CSP solution is obtained by sequentially building patterns through subroutine *getPattern*: the subroutine attempts to solve subproblems where the sum of prices is maximized under bounded knapsack and compatibility constraints. For each configuration $\mathbf{c} = \langle$stock type, first cut orientation$\rangle$ in the set C of all possible configurations, a cutting pattern is generated and added to a set P. The most profitable pattern $a \in P$ is then selected using *getBestPattern*, which also set the corresponding activation level *cuts*. The *fillingPattern* function is implemented to insert unplaced items in empty sections, given the residual demand and the value of *cuts* of the chosen pattern. Subsequently, prices λ_i are updated by promoting items with high current demand d'_i and small multiplicity in a. A complete CSP solution s is finally obtained by combining the partial solution computed for demand $\mathbf{d}'$ with the one found for the residual demand $\mathbf{r}'$.

In general, open stacks are not an objective to be minimized, but are limited by the number of unloading stations that equip the cutting machine. Although the pattern generation procedure already respects the limited availability of unloading stations, an attempt at reducing the stacks opened by the current solution s is made by the *patternSeq* procedure. The function *patternSeq* implements a simple pattern sequencing heuristic that compares each pattern with all the subsequent ones, and swaps pattern pairs that allow the largest decrease of open stacks.

The current solution is added to a pool (*Sol*) if it fulfills all the constraints and is not dominated by any other. The process is iterated a prescribed number N of times unless certain halting conditions hold (e.g., user stop or minimum used area reached). Finally, the pool *Sol* is filtered and reduced to a frontier of non-dominated solutions.

It may happen that *Sol* consists of partial solutions only, due to insufficient stock availability. In that case, non dominated solutions are limited to those that fulfills the largest demand.

For the sake of conciseness, in the following we shall assume $\mathbf{c} = \langle j, H \rangle$, i.e., the cut of the first stage is always parallel to the height H_j of the stock item: the arguments presented maintain validity when replacing "horizontal" with "vertical" and "H_j" with "L_j".

3.2 Patterns Generation and Selection

For a given configuration $\mathbf{c} = \langle j, H \rangle \in C$, function *getPattern* creates a pattern that depends on the current prices λ and demands $\mathbf{d}'$. First, for each part type $i \in I'$ a vertical section of width l_i is defined, and a bounded integer knapsack problem is solved. The knapsack capacity is set equal to the stock height H_j and element sizes correspond to the heights h_i of the part types selected. Clearly, only part types $k \in I'$ with $l_k \leq l_i$, demand $d'_k > 0$ and prices $\lambda_k > 0$ are considered. Actually, each knapsack element describes a horizontal *strip* containing a single item k (2-stage patterns), or multiple copies of the same part-type (3-stage patterns) wide at most l_i. In the latter case, the maximum multiplicity m_k is given by $\lfloor l_i / l_k \rfloor$, and prices are computed accordingly.

Let Q_c be the set of sections generated so far, each one provided with a certain activation level defined with respect to the current part type demand $\mathbf{d}'$. The pattern is obtained by solving a further bounded knapsack problem that optimally selects sections in Q_c. In particular, the knapsack capacity is equal to L_j and knapsack elements correspond to sections, where the size and value of element q are respectively equal to the width and price of section q (the price of a section is the sum of the prices of items it contains). The resulting solution provides a subset of sections $\bar{Q}_c$ that may present overproduction. In order to preserve the structure of the knapsack problem, only activation levels of the elements (sections) are explicitly considered, whereas part type multiplicities in sections are neglected; moreover, the simultaneous presence of part types and rotated part types is not accounted for. It follows that part types overproduction is not prevented and a post-processing may be required to exactly fulfill part type demands. Sections $q \in \bar{Q}_c$ are sorted by non-increasing order of the ratio $\frac{V_q}{S_q}$, where V_q and S_q are respectively the price and the area of section q.

Sections are possibly added to the pattern p following the prescribed order, and demand $\mathbf{d}'$ is accordingly updated. In particular, a section is discarded if it causes demand overproduction, or overflow of the allowed limit on open stacks. In that case, the resulting pattern p may present a residual stock area of length R_j that can be further filled. Therefore the whole procedure, i.e., sections generation and selection, is repeated in order to generate a sub-pattern for the configuration $\mathbf{c} = \langle \bar{j}, H \rangle \in C$, where stock item $\bar{j}$ has length R_j and height H_j. The sub-pattern is placed next to p and merged with it, and the process reiterated until the residual stock area cannot be filled any longer. Pattern p is then added to the set P.

Once patterns are generated for each $\mathbf{c} \in C$, the procedure *getBestPattern* selects the most profitable pattern $a \in P$, that is, a pattern that maximizes the total price of the sections it contains. Pattern a is finally added to the partial CSP solution s with activation level *cuts*, computed according to the current demand $\mathbf{d}'$ and stock items availability. The pattern generation process is iterated until the whole demand is fulfilled.

Since the presence of real-valued prices, all the above bounded integer knapsack are solved by encoding the integer variables as binary variables and then by using an own implementation of the algorithm for the 0-1 knapsack described in (Martello and Toth, 1990).

3.2.1 Precut Policy

For each pattern p, function *getPattern* generates also a pattern p' that contains a *precut*, and then evaluates the opportunity of replacing p with p'. For a given pattern p, the algorithm scans all the cuts of the second stage (those required to separate strips), choosing one whose quota h is closest to the middle of H_j, see Figure 1.

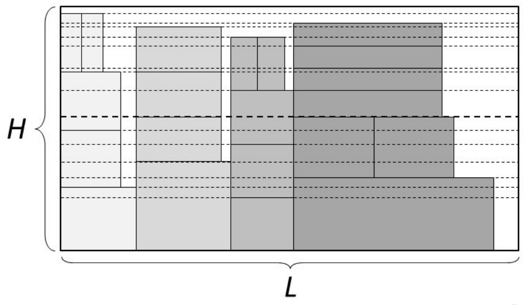

Figure 1: Dotted lines indicate the potential cuts of the second stage. The bold line refers to the cut closest to the middle of the stock height H_j.

Stock item j (and pattern p) is split into new stock items j_l and j_u (sub-patterns p_l and p_u) of sizes $h \times L_j$ and $(H_j - h) \times L_j$, respectively; the items that cross quota h are removed and the sub-pattern p_u deleted, see Figure 2. Stock item j_u is used to build a new pattern p'_u (by the same procedure described above), and a pattern with precut p' is obtained by merging p'_u and p_l, see Figure 3. If the difference between the reduced costs of p and p' is larger than a given threshold δ, then the pattern with precut is selected; otherwise *getPattern* selects p.

The value of δ is set to zero in Steps 1 and 2 of the heuristic, and is progressively increased in Step 3, where the algorithm aims at producing solutions with a decreasing number of precuts.

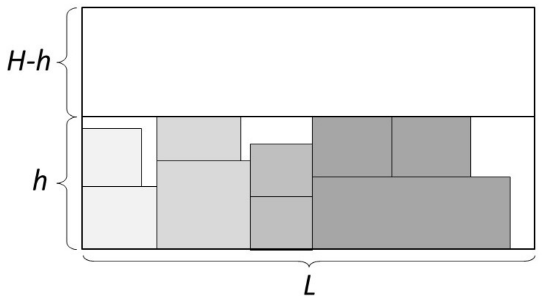

Figure 2: Sub-pattern p_l.

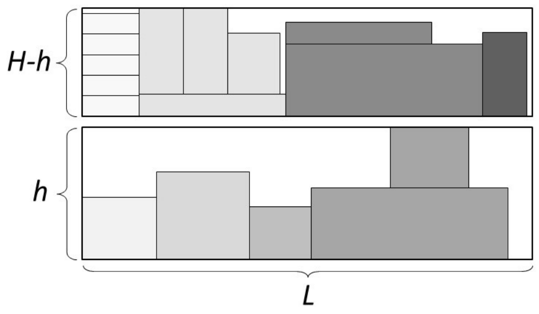

Figure 3: Sub-patterns p_l and p'_u.

4 COMPUTATIONAL RESULTS

The SVC was implemented in C++ and experiments were performed on a Intel® Core2 Duo E8500 3.16 GHz with 8Gb RAM. The SVC heuristic has been tested on ten real instances provided by SCM Group, with $n \in [58, 972]$ and $\rho = 1.15$. The results have been compared to those provided by the software previously adopted by the company and developed by a national software house devoted to the design of cutting optimization software. In order to be consistent, the non-dominated solution using the minimum total stock area has been promoted to the *best* solution and compared to the single solution provided by the company own software.

Table 1 and 2 show the results for the case with disabled and allowed precut, respectively. For each instance, the SVC performance is evaluated in terms of total used stock area (Area), number of used stocks (#stocks), total number of patterns (#patterns) and CPU time. The last three columns show the percentage gaps with the results obtained by the company software (positive values mean an improvement obtained by SVC; the symbol '-' means that the CPU time limit of three hours has been reached by the company software).

On average, the SVC heuristic reduces the total used stock area by 2.34% without precuts (1.42% with precuts), while the percentage trim-loss is reduced by a factor between 1.5 and 4. The number of used stocks is also reduced by 7.62% (1.19 %) on the mean. When precut is disabled, an average reduction of 3.61% in the number of distinct patterns is achieved, although a meaningful mean increase of 11.02% arises when precut is allowed. However, such worsening is mainly due to the greater material usage efficiency attained by the SVC *best* solution: in most of the cases, SVC computes non-dominated solutions requiring comparable used material and less number of patterns. The CPU time required to solve real in-

Table 1: Results without precut on ten real instances.

		SVC heuristic				Company software		
Name	*n*	Area	#stocks	#patterns	CPU time	ΔArea (%)	Δstocks (%)	Δpatterns (%)
I_1	58	2403.12	304	69	26.69	1.31	1.32	2.90
I_2	83	1454.8	251	102	6.67	1.19	1.20	-6.86
I_3	141	1512.21	508	165	121.14	1.58	1.57	0.61
I_4	500	221.32	29	29	103.52	2.40	34.48	34.48
I_5	500	273.84	36	36	-	-	-	-
I_6	747	3158.81	620	546	634.72	2.13	1.94	-14.65
I_7	759	495.92	96	96	760.08	5.45	5.21	5.21
I_8	869	2923.78	563	269	972.06	-	-	-
I_9	951	354.72	67	67	34.63	-	-	-
I_10	972	903.11	179	178	494.59	-	-	-
Average	**-**	**1370.16**	**265.30**	**155.70**	**350.46**	**2.34**	**7.62**	**3.61**

Table 2: Results with precut on ten real instances.

		SVC heuristic				Company software		
Name	*n*	Area	#stocks	#patterns	CPU time	ΔArea (%)	Δstocks (%)	Δpatterns (%)
I_1	58	2403.12	304	80	5.92	1.64	1.64	-18.75
I_2	83	1454.8	251	105	34.55	0.79	0.80	-6.67
I_3	141	1509.24	507	152	332.28	1.18	1.18	-8.55
I_4	500	220.78	29	29	139.09	-	-	-
I_5	500	273.3	36	36	471.14	-	-	-
I_6	747	3158.27	620	524	632.56	2.08	1.13	-10.11
I_7	759	494.83	96	96	104.28	-	-	-
I_8	869	2908.4	566	264	549.39	-	-	-
I_9	951	354.72	67	67	84.11	-	-	-
I_10	972	903.33	179	177	1332.55	-	-	-
Average	**-**	**1368.08**	**265.50**	**153**	**368.587**	**1.42**	**1.19**	**-11.02**

stances is about 6 minutes on the mean, with just few instances that exceed 10 minutes.

The heuristic was also tested on 52 benchmark instances with $n \in [10, 250]$. With the precut option disabled (enabled) the algorithm reduces trim-loss in 17.3% (15.4%) of the cases. By using the continuous lower bound of the total requested area, the optimality of the best solution with respect to the material usage has been certified in the 80.8% (82.7%) of the cases. Independently of the precut policy, in 17.3% of the cases a solution with one pattern less is found, and only 1.9% of the instances require an additional pattern.

5 CONCLUSIONS AND FUTURE WORK

Mathematical programming is a powerful tool that can be used to model many relevant industrial problems. However, those mathematical formulations are often challenging and, in some cases, real process constraints are hard to be modeled. This is also true for many CSP variants that try to simplify resolution by encoding just a restricted number of real-world constraints. Software applications based on those CSP formulations may then happen to be inappropriate, and companies are continuously looking for tools able to deal with the whole process features and specification.

This paper addressed such an issue and presented an SVC heuristic to solve a rich M2D-CSP that fulfills the specification of a family of wood cutting machines produced by *SCM Group*. The objectives of used stock area minimization and of additional criteria, as well as the fulfillment of open stack constraints, were considered. Computational tests demonstrated the improvement achieved on the previous software solution, in terms of both solution quality and CPU time requirements.

Further research is needed to include additional optimization criteria, e.g., the minimization of cutting times and the management of left-overs, and to provide an exact formulation of the pricing problem when the whole set of specification is considered. Also, cutting processes are strongly affected by operations scheduling: the formulation of a bi-objective rich M2D-CSP that simultaneously minimizes a scheduling function (such as the maximum lateness or the weighted sum of tardiness) should then deserve investigation.

ACKNOWLEDGEMENTS

We gratefully acknowledge dr. eng. Federico Ratti, Stefano Oliva and Simone Cera of SCM Group for the fruitful discussions and constructive remarks that helped us design and refine the algorithm described in this paper.

REFERENCES

Aloisio, A., Arbib, C., and Marinelli, F. (2011). On lp relaxations for the pattern minimization problem. *Networks*, 57(3):247–253.

Arbib, C., Marinelli, F., and Ventura, P. (2016). One-dimensional cutting stock with a limited number of open stacks: bounds and solutions from a new integer linear programming model. *International Transactions in Operational Research*, 23(1-2):47–63.

Belov, G. and Scheithauer, G. (2002). A cutting plane algorithm for the one-dimensional cutting stock problem with multiple stock lengths. *European Journal of Operational Research*, 141(2):274 – 294.

Belov, G. and Scheithauer, G. (2007). Setup and open-stacks minimization in one-dimensional stock cutting. *INFORMS J. on Computing*, 19(1):27–35.

Chen, Q., Cui, Y., and Chen, Y. (2016). Sequential value correction heuristic for the two-dimensional cutting stock problem with three-staged homogenous patterns. *Optimization Methods and Software*, 31(1):68–87.

Chu, C. and Antonio, J. (1999). Approximation algorithms to solve real-life multicriteria cutting stock problems. *Operations Research*, 47(4):495–508.

Cui, Y.-P., Cui, Y., and Tang, T. (2015). Sequential heuristic for the two-dimensional bin-packing problem. *European Journal of Operational Research*, 240(1):43 – 53.

Malaguti, E., Durn, R. M., and Toth, P. (2014). Approaches to real world two-dimensional cutting problems. *Omega*, 47:99 – 115.

Martello, S. and Toth, P. (1990). *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, Inc., New York, NY, USA.

Nonås, S. L. and Thorstenson, A. (2000). A combined cutting-stock and lot-sizing problem. *European Journal of Operational Research*, 120(2):327 – 342.

Poldi, K. and Arenales, M. (2009). Heuristics for the one-dimensional cutting stock problem with limited multiple stock lengths. *Computers and Operations Research*, 36(6):2074–2081.

SCM (2017). SCM group. https://www.scmgroup.com. Accessed: 2017-06-30.

Umetani, S., Yagiura, M., and Ibaraki, T. (2003). One-dimensional cutting stock problem to minimize the number of different patterns. *European Journal of Operational Research*, 146(2):388 – 402.

Varela, R., Vela, C. R., Puente, J., Sierra, M., and González-Rodríguez, I. (2008). An effective solution for a real cutting stock problem in manufacturing plastic rolls. *Annals of Operations Research*, 166(1):125.

Wäscher, G., Haußner, H., and Schumann, H. (2007). An improved typology of cutting and packing problems. *European Journal of Operational Research*, 183(3):1109 – 1130.

Yanasse, H. H. and Lamosa, M. J. P. (2007). An integrated cutting stock and sequencing problem. *European Journal of Operational Research*, 183(3):1353 – 1370.

Yanasse, H. H., Zinober, A. S. I., and Harris, R. G. (1991). Two-dimensional cutting stock with multiple stock sizes. *Journal of the Operational Research Society*, 42(8):673–683.