UNIVERSITÄT OSNABRÜCK

Cutting Stock by Iterated Matching

Andreas Fritsch, Oliver Vornberger

University of Osnabrück
Dept. of Math./Computer Science
D-49069 Osnabrück
andy@informatik.uni-osnabrueck.de

Abstract

The combinatorial optimization problem considered in this paper is a special two dimensional cutting stock problem arising in the wood, metal and glass industry. Given a demand of non oriented small rectangles and a theoretically infinite set of large stock rectangles of given lengths and widths, the aim is to generate layouts with minimal waste, specifying how to cut the demand out of the stock rectangles. Special restrictions are chosen with respect to the production conditions of the flat glass industry. An iterative algorithm for solving this problem heuristically is developed. By a maximum weight matching suitable rectangles are matched in every iteration step and the matched pairs are considered as new, so called meta rectangles. These meta rectangles can be treated in the same way as ordinary rectangles. Furthermore, by the use of shape functions the orientations of the demand rectangles are not fixed until the layout has been computed. The algorithm, programmed in C, has been tested with several instances, containing between 52 and 161 rectangles, taken from real demands of a glass factory. The resulting layouts, calculated within a few minutes (on a PC-486), have an average waste of less than 6 percent.

1 Introduction

Because of the increasing costs of raw material and the need to avoid industrial waste, solving cutting stock problems became of great interest in the area of operations research. Cutting stock problems belong to the class of combinatorial optimization problems, so a solution among all possible solutions has to be found, which optimizes a criterion function subject to a set of constraints. Unfortunately the size of the solution space containing all feasible solutions in general is enormous. As a consequence an exhaustive search within the whole solution space is impractical and sophisticated heuristics which ensure the calculation of reasonable solutions within limited time are required.

The combinatorial optimization problem considered in this paper is a special two dimensional cutting stock problem arising in the wood, metal and glass industry. Given a demand of small rectangles and a set of large stock rectangles, layouts have to be found which specify how to cut the demand out of the stock rectangles with minimal waste.

Gilmore and Gomory used linear programming to solve such kind of a problem exactly [Gil 1961] [Gil 1965]. Their suggestions were improved by Herz [Her 1972]. Branch&Bound and tree search algorithms were developed by Cani [Can 1979] and Whitlock & Christofides [Whi 1977]. These algorithms fail if more than 20 rectangles have to be packed. Most of the heuristic algorithms for generating layouts are based on a greedy strategy. After sorting the demand rectangles they are placed in the stock rectangles and none of them can be repositioned. Level oriented packing algorithms were developed by Coffman, Garey, Johnson and Trajan [Cof 1980]. These algorithms are fast, but the performance bounds are not good enough to stand the requirements of the industry [Cof 1990] [Gar 1981].

A weakness of the most known algorithms is that at the beginning of the algorithm the alignments of all rectangles that have to be packed are fixed. In many domains, especially in the flat glass industry, a fixing of the alignment is unnecessary and it only leads to the fact, that some layouts with probably minimal waste are not considered. Furthermore, the most algorithms are unable to pack a high number of demand rectangles as dense as it is necessary for the flat glass industry. Therefore, an algorithm is needed which is able to handle a set of many rectangles (up to 200) and which takes every possible and suitable layout in consideration.

In the second section of this paper a detailed description of the considered problem is given and the solution strategy is presented. Since shape functions are used for solving the problem, an introduction to this theme is given in section three. The fourth section deals with slicing trees. In the fifth section the maximum weight matching problem is described. The developed algorithm consists of four stages which are described in section six. Computational results are given in section seven.

2 The Two Dimensional Cutting Stock Problem

The problem considered in this paper can be formulated as follows:

> Given a finite number n of demand rectangles of length l_i and width w_i and a theoretically infinite set of stock rectangles $\{R_1, R_2, \ldots\}$ of a given length and width. Find layouts for cutting all demand out of the stock rectangles so that the minimum number of stock rectangles is required. Only guillotine cuts are permitted. A rotation of the demand rectangles of $90°$ is allowed.

The above constraints are chosen with respect to the production conditions in the flat glass industry. A $90°$ rotation of the rectangles is possible because glass is an isotropic material that means that it has the same structure in every dimension. Only guillotine cuts are permitted means that the cuts must go from one edge of a rectangle to the opposite edge in a straight line. This constraint is crucial, because most factories in the glass industry use a cutting machine which can only carry out these cuts. It is assumed that the stock rectangle is large enough, so that every demand rectangle fits in the stock rectangle in at least one orientation. Without loss of generality it is assumed that the lengths and widths of all rectangles are integer.

There are two more restrictions concerning the practicability of cutting in the glass industry: the construction of traverses and the observance of cut distances. For cutting the demand rectangles out of the stock rectangles the latter are placed on a cutting table. There a carriage with a diamond placed at the bottom of it slices the glass sheet in x- and y-dimension. Since the big glass sheets are even worse to handle (the typical extensions are up to six meters in length and three meters in width) the first cuts that have to be made are in y-direction and go directly from one point of the longer side of the sheet to the opposite side. They divide the stock rectangle into some sections. We call these cuts traverse cuts and the resulting sections of the stock rectangles are called traverses. Notice that the traverse cuts are also guillotine cuts. The x-dimensions of the traverses should not exceed a given value. For example, a typical bound of the traverse lengths is 2300 mm if a stock sheet of 6000 mm length is used. Since the maximal traverse length depends on the dimensions of the used stock and that of the used cutting table, this value should be given together with the other parameters of an instance like the dimensions of stock and demand rectangles.

A nice property of the slicing of glass is that no sawdust arise as it does by the cutting of wood. But it is crucial to take care of the distances between the cuts. The cutting of glass is done by slicing the stuff and then breaking it along the slice. If two slices are too close, the glass can not be broken. The minimal distance between two cuts is the double sheet thickness, but for a glass sheet of 4 mm thickness the typical value of 20 mm is chosen. Also this value should be given in the instance. Figure 1 shows how a stock rectangle can be sliced into the demand rectangles. The numbers give information about a possible sequence of the cuts. The first two cuts are traverse cuts which divide the glass sheet into three traverses.

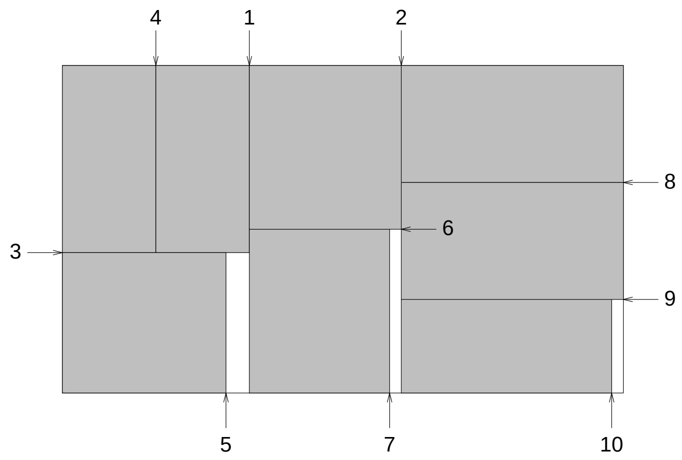


Figure 1: Possible guillotine cuts in a layout

For solving the given two dimensional cutting stock problem of the flat glass industry an iterative algorithm is developed based on the following idea: In every iteration step some rectangles are paired by a maximum weight matching and the matched pairs are considered as new, larger rectangles, so called meta rectangles. In the following iteration steps these meta rectangles are treated in the same way as ordinary rectangles. The iterative matching automatically takes care of the guillotine structure. The iteration stops if the constructed meta rectangles are traverses. These traverses will be assigned to the stock sheets by a first fit decreasing algorithm. In every iteration step all possible alignments of demand and meta rectangles are considered by making use of shape functions.

3 Shape Functions

A meta rectangle has several layouts according to the orientation of the horizontal or vertical cut which is used to split the meta rectangle into the two buddies it resulted from. The layout also depends on the alignments of the demand rectangles the meta rectangle contains. The alignments of the demand rectangles are not fixed at any time the algorithm proceeds. For example, if we match two demand rectangles A and B the resulting meta rectangle AB has eight essentially different layouts: A lies above or left from B and both rectangles can be rotated. Figure 2 shows the eight essentially different layouts for a meta rectangle consisting of two demand rectangles.

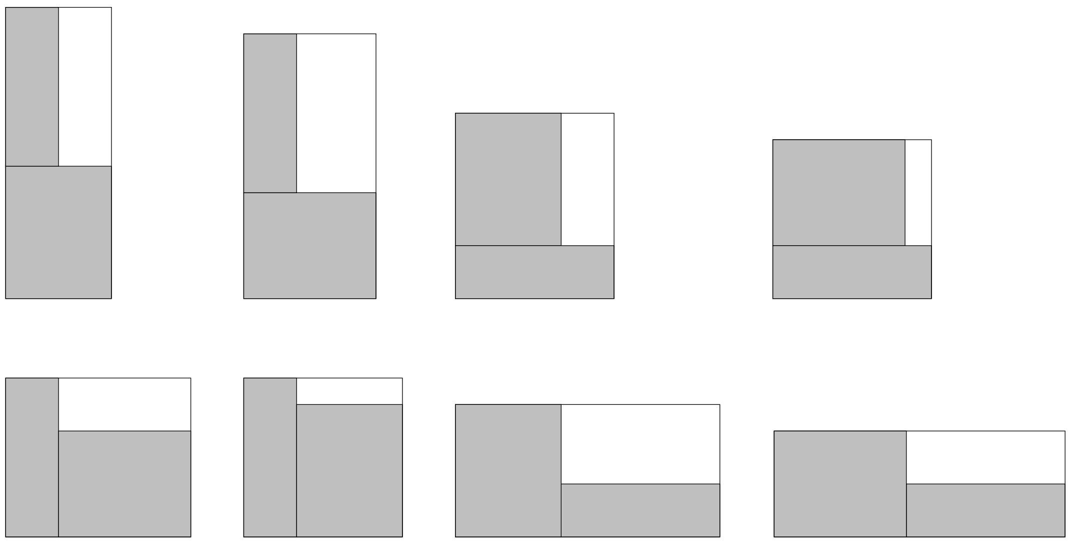


Figure 2: Layouts for a meta rectangle with two demand rectangles

A single layout of a meta rectangle is represented by a slicing instruction. This is a four dimensional vector (x, y, o, p), where x and y are the length and width of the meta rectangle in its considered layout, $o \in \{h, v\}$ is the orientation of the horizontal or vertical cut used to split it and p is the point (according to o) where the cut starts.

To describe all possible layouts of a meta rectangle, a shape function is used. Such functions are known from floorplanning problems [Sto 1983], [Ott 1983]. A shape function s_R for a rectangle R can be considered as a decreasing, piecewise linear function. The interpretation is that $s_R(w)$ is an upper bound on the length of a (meta) rectangle that has the width w. So, if a meta rectangle has been computed which contains a set of demand rectangles and if the dimensions of the stock rectangle are known, the maximal needed x-dimension for a layout of the meta rectangle can be found subject to the given width of the stock rectangle.

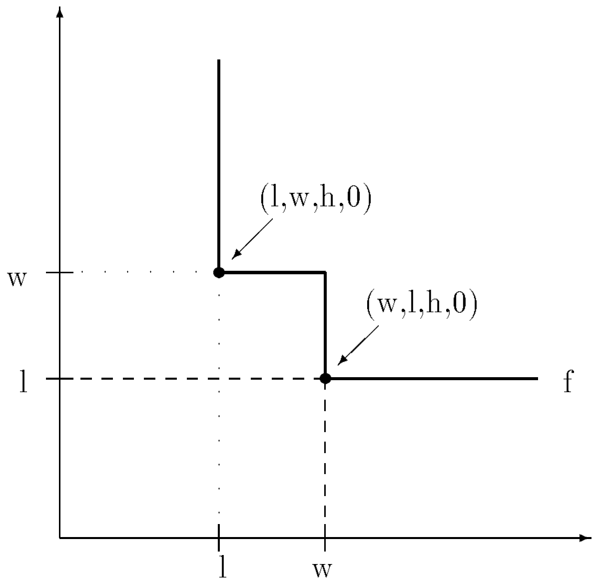


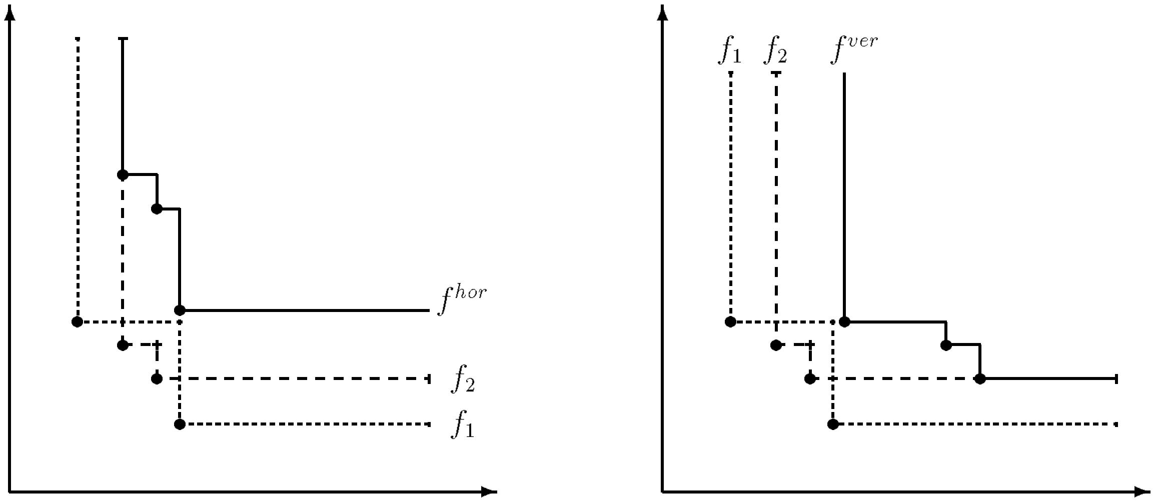
Figure 3: Shape function of a demand rectangle

A shape function is considered as a list of slicing instructions. The shape function of a demand rectangle of length l and width $w > l$ consists of two slicing instructions $((l, w, h, 0), (w, l, h, 0))$. The cut orientation and its starting point is of no use here, but later we will need this information for the cutting of meta rectangles unequal to demand rectangles. In Figure 3 an example of a shape function for a demand rectangle is given.

The shape function of a meta rectangle is calculated by the shape functions of the two demand or meta rectangles it consists of, with the help of the composition procedure: Let f_1 and f_2 be two shape functions and $\mathrm{s}^i = (x^i, y^i, o^i, p^i)$ a slicing instruction of f_i $(i = 1, 2)$. The composed slicing instruction for a horizontal resp. vertical cut is:

- horizontal cut: $\mathrm{comp}(\mathrm{hor}, \mathrm{s}^1, \mathrm{s}^2) := (\max(x^1, x^2), y^1 + y^2, h, \min(y^1, y^2))$
- vertical cut: $\mathrm{comp}(\mathrm{ver}, \mathrm{s}^1, \mathrm{s}^2) := (x^1 + x^2, \max(y^1, y^2), v, \min(x^1, x^2))$

By composing all slicing instructions of f_1 with those of f_2, two shape functions f^{hor} and f^{ver} are computed which contain all possible horizontal resp. vertical slicing instructions. Figure 4 shows a horizontal and vertical shape function, resulting from the composition of the shape functions of two demand rectangles. Three slicing instructions per cut orientation are found. This leads to the fact that a layout of the resulting meta rectangle is not included in the list of slicing instructions, if another layout of the same width but a smaller length or the same length but a smaller width exists. Hence, only feasible slicing instructions are stored in the shape function. For example, layout three and four in the upper row and layouts one and two in the lower row of Figure 2 are treated in this way.



a) horizontal b) vertical

Figure 4: Shape functions with given cut orientation

To calculate a single shape function f for a meta rectangle, we take the minimum about f^{hor} and f^{ver} by merging this two functions and deleting the dominated slicing instructions. For example, a slicing instruction of f^{ver} is dominated by one of f^{hor}, if $f^{hor}(w) < f^{ver}(w)$ for a given width w. If two slicing instructions of f^{hor} and f^{ver} have the same width and the same length then we will determine here, that the horizontal cut is chosen.
The maximal number of slicing instructions that have to be stored in a shape function f which is calculated by composing two other shape functions f_1 and f_2 is $2 \cdot (|f_1| + |f_2| - 1)$, where $|f_i|$ denotes the number of slicing instructions in function f_i (see [Mue 1991]).

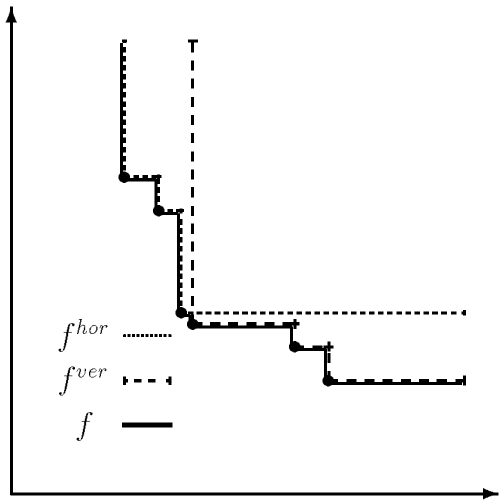


Figure 5: Minimum shape function

Figure 5 shows the minimum shape function of the functions f^{hor} and f^{ver} shown in Figure 4. In this example of calculating the minimum function, no dominated slicing instructions have to be deleted.

4 Slicing Trees

To store the actual pairs in every iteration step we use binary slicing trees. Every leaf of a tree represents a demand rectangle, an interior node of a tree represents a meta rectangle. Two nodes have the same father if the rectangles represented by these nodes are paired. In every node the shape function of the corresponding meta rectangle is stored.

A slicing tree is built bottom up. At the beginning of the algorithm all slicing trees consist of only one node (the root) representing the demand rectangles. In every iteration step, some of the trees are combined by a common father, which is the root of a new slicing tree representing the new meta rectangle. The iteration stops, if no more suitable pairs can be matched. A detailed description of this stop criteria is given in section six. Since all nodes contain a shape function and every shape function describes all possible layouts of a meta rectangle, the slicing trees contain an exponential number of possible layouts for the stock rectangles.

When the algorithm has ended, we can detect how the demand rectangles should be cut out of the stock rectangles. Therefore, a tree is traversed in top down order. Since every node stores a shape function we can detect how the first cut in the stock rectangle has to be made by considering the shape function in the root of a tree and searching for the first slicing instruction whose y-value is smaller or equal to the width of the stock rectangle. Due to the property of the shape functions mentioned above, we automatically get the slicing instruction which describes the layout with the minimal length according to the made pairings. If the first cut has been carried out, the stock rectangle is split into two parts. The shape functions in the sons of the root describe how these two parts efficiently have to be split further. This proceeding is shown in Figure 6 by an example. By the traversal of all slicing trees (one slicing tree for every layout) the demand rectangles can be produced.

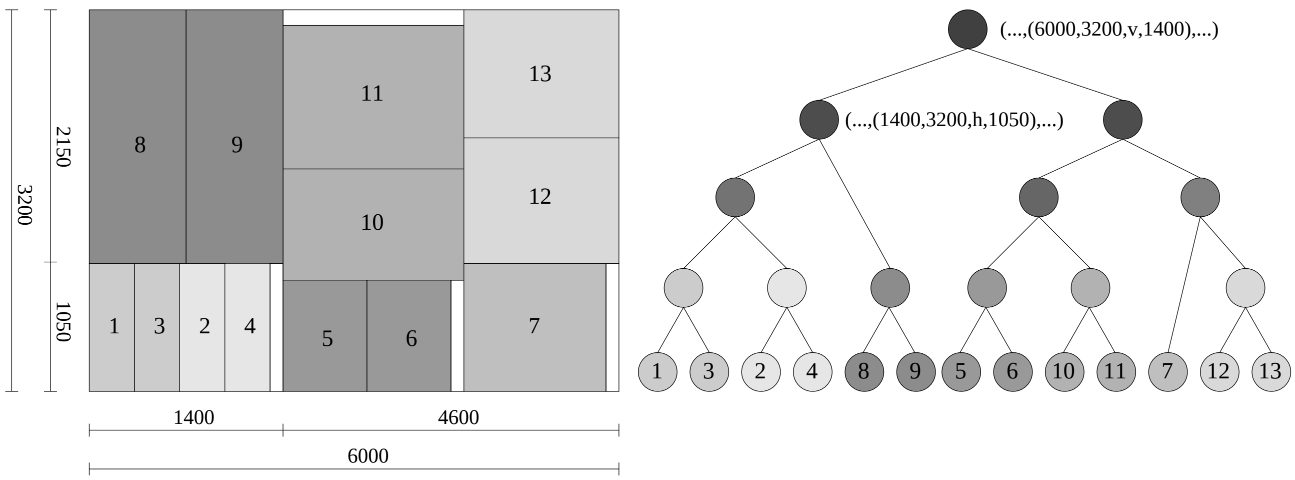


Figure 6: Layout and corresponding slicing tree

5 Maximum Weight Matching

To find out in every iteration step, which of the available rectangles have to be paired, a maximum weight matching in a complete undirected weighted graph is used. The nodes of the graph represent the available (meta) rectangles, the edges represent the possible pairs and the weights give information about the benefit of a pair. The higher the weight, the better the pair. The weights are found by a so called benefit function, which evaluates the possible pairing of two rectangles by considering the corresponding shape functions. For example, a possible benefit function for the pairing of two rectangles is to calculate the average waste of all possible layouts of the resulting meta rectangle and to subtract this value from 100. This encourages the formation of meta rectangles having many densely packed layouts. If all weights of the edges are calculated, the maximum weight matching finds the set of edges, which maximizes the sum of all weights under the constraint that no two edges are adjacent.

This problem can be formulated as a LP-problem as follows:

Maximize

$$\sum_{i,j=1}^{n} c_{i,j} \cdot x_{i,j}$$

subject to

$$\begin{aligned} \sum_{j=1}^{n} x_{i,j} &\leq 1 && \text{for} && i = 1,\ldots,n \\ x_{i,i} &= 0 && \text{for} && i = 1,\ldots,n \\ x_{i,j} &= x_{j,i} && \text{for} && i,j = 1,\ldots,n \\ x_{i,j} &\in \{0,1\} && \text{for} && 1 \leq i \leq j \leq n \end{aligned}$$

Here, n is the number of available meta rectangles and $\mathbf{c}$ is the cost matrix. The value of $x_{i,j}$ is 1 if the rectangles i and j are matched and $x_{i,j} = 0$ if not.

Edmonds showed, that the maximum weight matching problem can be solved in time $O(n^3)$. The algorithm used in this application was suggested by Gabow [Gab 1973]. Figure 7 shows the matching algorithm for seven rectangles in three iteration steps. The edges with zero benefit are not shown.

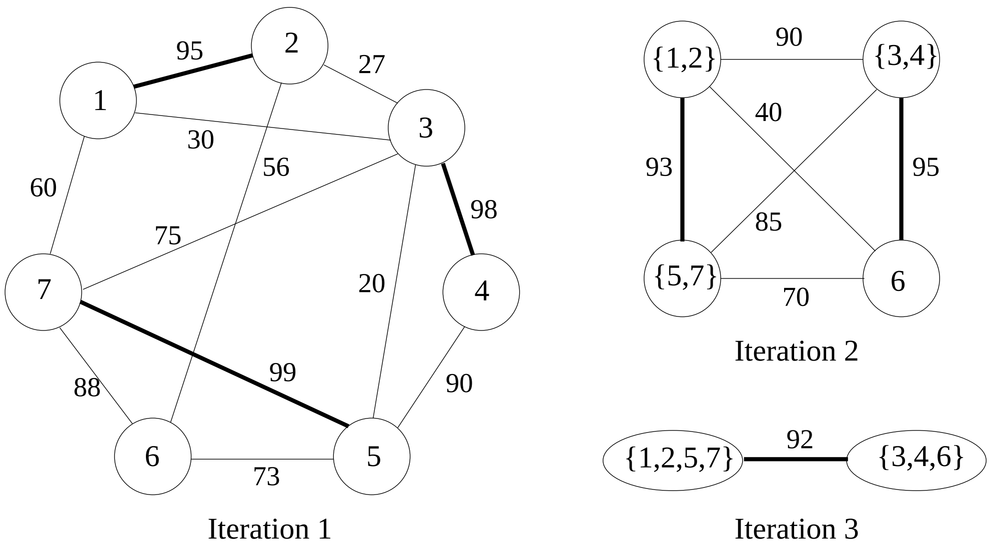


Figure 7: Maximum weight matching in three iterations

6 Iterated Matching

Up to now, it has not been mentioned at which time the iteration stops and what a suitable pair of rectangles is.

The stop criteria is chosen as follows: One restriction when cutting glass sheets is, that the stock rectangles must be split into traverses at the beginning of the cutting; the first cuts must be parallel to the shorter side of the stock rectangle, so that the stock rectangle is divided into sections with the same width as and a smaller length than the original sheet. So the iterated matching stops if meta rectangles have been generated with at least one layout in which they represent a traverse. By computing such meta rectangles the problem of finding the layouts for the stock rectangles is reduced to a one dimensional bin packing problem: the traverses, all of nearly the same width as that of the stock rectangles, have to be assigned to the latter in such a way, that the stock sheets are filled nearly complete in length.

To find only suitable meta rectangles the algorithm has to take care of the further restrictions as it is the bound of the traverse lengths, the cut distances and the observance of the guillotine structure. The last is preserved since the iterated matching automatically takes care of the guillotine structure as seen above. To prevent cuts with distances of less than the given minimum, slicing instructions which do not fulfill this subjection are not stored during the computation of shape functions. To reduce the number of the slicing instructions of a shape function also those instructions are not stored which have a x- or y-value bigger than the larger side of the stock rectangle. The benefit functions estimate a potential pair with zero value if the resulting meta rectangle does not have a layout in which it fits into the given traverse shape. All these observations are made in any iteration step.

The developed algorithm consists of four stages. The first two stages are used to construct traverses. In the third stage the traverses will be improved. The fourth stage is needed to align the traverses to the stock rectangles.

In the first stage universal rectangles are calculated by an iterative use of the maximum weight matching. To call a meta rectangle universal, it must comply with two requirements: 1. The average waste of all layouts of the meta rectangle must be smaller than five percent and 2. There must exist another meta rectangle, so that the meta rectangle resulting from the pairing of these two is a traverse. A traverse should have a waste area of less than 4 percent. In every iteration step of stage one a matching graph is formed which nodes represent the available non universal rectangles. The benefit function in stage one delivers a high value for a possible pair if the average percentage waste of all layouts of the potentially resulting meta rectangle is smaller than five percent. The benefit is zero if all layouts of the resulting meta rectangle violate the restrictions like traverse lengths and cut distances. After the calculation of all weights in the matching graph the maximum weight matching chooses the best marriages. Stage one ends if no more universal rectangles can be constructed without violating the restrictions.

The second stage consists of only one iteration step. Here the n constructed universal rectangles are matched to $n/2$ traverses. For this also the maximum weight matching is used. As an indicator for the decision whether a meta rectangle (resulted from the pairing of two universal rectangles) is a traverse or not, the percentage of the used area of its "relevant" layout is computed. To find the relevant layout, the first slicing instruction of the shape function is chosen with a y-value smaller than or equal to the width of the stock rectangle. By multiplying the x- and the y-value of this slicing instruction the area which is claimed by the corresponding layout is calculated. To compute the percentage of the used area, only the areas of all contained demand rectangles have to be added, multiplied with 100 and divided by the claimed area. The higher the percentage of the used area, the better the traverse. This has to be respected by the benefit function of stage two, also ensuring that none of the other restrictions is violated.

Since the iterated matching is a greedy strategy, the third stage is needed to improve those traverses with a lot of waste area. The strategy of improvement is to choose the worse traverses, for example those with less than 96 percent used area, and improve them by interchanging subtrees of the corresponding slicing trees. To find out whether an interchangement should be made or not, the sum of all x-dimensions of the traverses is calculated before and after the interchangement. The idea is that the traverses are improved, if the sum of all traverse lengths after interchanging is smaller than before it, because then all demand rectangles have been packed on a smaller domain. Hence less waste will be produced. Figure 8 shows this proceeding by an example.

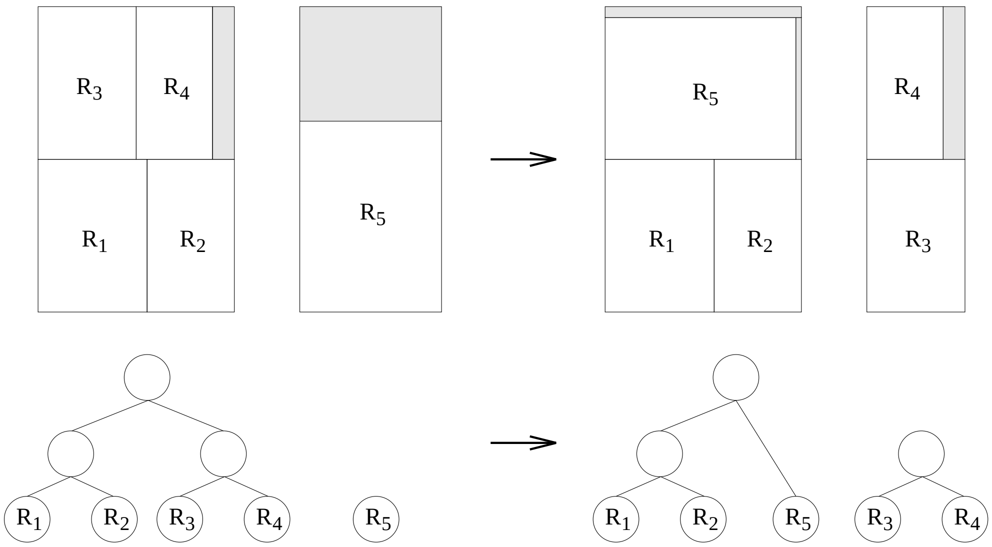


Figure 8: Improvement of traverses by interchanging subtrees

Since the traverses all have nearly the same width as the stock rectangle, the problem of packing the rectangles into the stock sheet is reduced to a one dimensional bin packing problem. Therefore, in the fourth stage the traverses are assigned to the stock rectangles. This could be done by a first fit decreasing algorithm: list the traverses ordered by decreasing x-dimension and let the stock sheets be indexed by increasing numbers. Assign the first traverse of the list to the stock rectangle with the lowest index it fits in. Then assign the second traverse by the same rule. If a traverse fits in none of the already used stock sheets, a new stock rectangle has to be used. The first fit decreasing finds solutions for this one dimensional bin packing which are at most 22 percent worse than the optimal solution, and it can unfortunately be this bad [Gar 1979]. Therefore a modified first fit decreasing algorithm is used. The modification is, that after the normal first fit decreasing is completed, a systematic interchange of traverses between the used stock rectangles is made with the aim to fill the stock rectangles as full as possible. After the assignment has been computed, all traverses of one stock sheet are paired and the corresponding slicing trees and shape functions are computed. So, at the end of stage four for every used stock rectangle a slicing tree is computed which gives information about how to cut the glass sheet. The good results of the stock cutting by iterated matching are shown in the tables of the following section.

7 Computational Results

The algorithm, programmed in C, has been tested with several instances taken from real demands of a glass factory, containing between 52 and 161 rectangles. A PC-486 (33 MHz) was used to measure the time requirements of the algorithm. The time give information about the real computation time, i.e. requirements for input and output are not considered. The stock rectangles are of dimension about 6000 mm x 3200 mm for all instances. The maximal traverse length is 2300 mm and the minimal cut distance is 20 mm for all instances except instance three. Here the minimal cut distance is 40 mm since the thickness of the required stock sheet is 6 mm and not 4 mm as it is for the other instances.

Results of the Iterated Matching Algorithm

instance	n	area in mm^2	time in sec.		average waste in %			LNS	RNS
			stage 1+2	stage 3	match	traverse	stock		
1	116	131309371	50	42	6.58	4.96	5.46	7.313	7.351
2	161	124828209	235	1555	9.11	5.18	5.34	6.968	6.980
3	52	58237035	30	82	16.05	8.35	9.73	3.396	3.448
4	131	100206712	69	95	5.33	4.59	4.60	5.559	5.560
5	88	76925672	31	25	7.46	4.80	4.84	4.277	4.279
6	157	159191327	92	70	5.73	3.87	3.97	8.765	8.774
7	91	110356737	34	31	6.03	4.82	6.40	6.137	6.241

Table 1: Results of the four stage algorithm

Table 1 shows the data of the instances and the computational results of the four stage algorithm discussed in section six. The number n of the demand rectangles of the instances is given in the second column and the sum of all demand rectangle area is listed in the third column. The time column gives information about the time requirements in seconds of stage one, two and three. Stage one and two are considered together, because they both show how fast the iterated matching works. The time needed to run the modified first fit decreasing is not listed, because it only takes less than one second. The match waste is the average waste in percent of all traverses after stage two, the traverse waste is the waste after stage three and the stock waste is the average waste of all computed layouts at the end of the algorithm.

The number of the at least needed stock rectangles (LNS) and of the real needed stock rectangles (RNS) gives information about how good the modified first fit decreasing assigns the traverses to the stock rectangles. LNS is a lower bound for the number of required stock rectangles, RNS is the fractional number of really needed stock rectangles. LNS and RNS are calculated as follows: Let k be the number of traverses after stage three, t_i the length of traverse i according to the given width W and length L of the stock sheet. Then

$$\text{LNS} = \frac{1}{L} \cdot \sum_{i=1}^{k} t_i$$

is a lower bound for the number of needed stock sheets. Notice, that it is not guaranteed that this bound can really be reached. To calculate RNS, let $s \in I\!N$ be the number of used stock rectangles $R_i, i = 1, \dots, s$. Assume that R_s is the stock sheet which is filled at least in length and let l denote the used x-dimension of R_s. Then

$$\text{RNS} = (s-1) + \frac{l}{L}$$

is the fractional number of really needed stock rectangles.

The iterated matching produces traverses with little average waste (about 8 percent) in a very short time. Exceptions are the instances two and three. Here the greedy behaviour of the matching strategy has to be adjusted. This is done in stage three.

The improvement of the traverses in stage three takes in most cases only a few seconds (no more than 95 seconds), except for instance two. If the third stage for solving instance two is considered in detail it can be seen that most of the improvement is done in the first thirty seconds. Hence, a time restriction for stage three would be useful. Notice, that it is not the number of demand rectangles, that makes the algorithm impracticable for instance two. Instance six involves nearly the same number of demand rectangles but it has the smallest waste of all computed instances and it takes only two minutes and 42 seconds to compute this solution.

The modified first fit decreasing finds good solutions for the assignment of the traverses to the stock rectangles. This can be seen in the very small difference between LNS and RNS. Analyzing Table 1, it can be seen that the average waste of all computed layouts is less than six percent and that the time requirement, with exception of instance two, is less than three minutes. Four of the seven instances really take less than 112 seconds for computation.

Hence, we can conclude that the iterated matching is a new efficient method for solving the two dimensional cutting stock problem subject to the restrictions of the industry which is applicable for instances with a large number of rectangles.

References

[Can 1979] P. de Cani, “Packing problems in theory and practice”, Department of Engineering Production, University of Birmingham, March 1979.

[Cof 1980] E. G. Coffman, M. R. Garey, D. S. Johnson and R. E. Tarjan, “Performance Bounds for Level-Oriented Two-Dimensional Packing Algorithms”, SIAM Journal on Computing 9, 4 (1980), pp. 808–826.

[Cof 1990] E. G. Coffman and P. W. Shor, “Average-Case Analysis of Cutting and Packing in Two Dimensions”, European Journal of Operational Research 44, 2 (1990), pp. 134–145.

[Gab 1973] H. Gabow, ”Implementation of Algorithms for Maximum Matching on Nonbipartite Graphs”, Ph.D. Thesis, Stanford University, 1973.

[Gar 1979] M.R. Garey und D.S. Johnson, "Computers and Intractability", Freeman, San Francisco, 1979, pp. 126–127.

[Gar 1981] M. R. Garey and D. S. Johnson, "Approximation Algorithms for Bin Packing Problems: A Survey", in Analysis and Design of Algorithms in Combinatorial Optimization, Vol. 266, G. Ausiello and N. Lucertini, eds., Springer Verlag, Berlin, 1981, pp. 147–172.

[Gil 1961] P. C. Gilmore and R. E. Gomory, "A Linear Programming Approach to the Cutting-Stock Problem", Operations Research, Vol. 9 (1961), pp. 849–859.

[Gil 1965] P. C. Gilmore and R. E. Gomory, "Multistage cutting stock problems of two and more dimensions", Operations Research, Vol. 13 (1965), pp. 94–120.

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

[Mue 1991] R. Müller, "Hierarchisches Floorplanning mit integrierter globaler Verdrahtung", Report No. 81, Fachbereich Mathematik–Informatik Universität–GH–Paderborn, February 1991, pp. 63–64.

[Ott 1983] R.H.J.M. Otten, "Efficient Floorplan Optimization", In Proceedings of the International Conference on Computer Design: VLSI in Computers, IEEE, 1983, pp. 499–502.

[Sto 1983] L. Stockmeyer, "Optimal Orientations of Cells in Slicing Floorplan Design", Information and Control, No. 57 (1983), pp. 91–101.

[Whi 1977] C. Whitlock and N. Christofides, "An Algorithm for Two-Dimensional Cutting Problems", Operations Research, Vol. 25, Nr. 1, January-February 1977, pp. 30–44.