FOUNDATIONS OF CONTROL ENGINEERING
Vol. 14 (1989) No. 4

TWO-DIMENSIONAL CUTTING PROBLEM BASIC COMPLEXITY RESULTS AND ALGORITHMS FOR IRREGULAR SHAPES

JACEK BŁAŻEWICZ*, MACIEJ DROZDOWSKI*, BOLESŁAW SONIEWICKI**, RAFAŁ WALKOWIAK*

Presented by J. Węglarz

Abstract. This paper deals with two-dimensional cutting problems. Firstly the complexity of the problem in question is estimated. Then, several known approaches for the regular (rectangular) and irregular (not necessarily rectangular) cutting problems are described.

1. INTRODUCTION

In this work we present basic results for two-dimensional cutting problem. This problem consists in cutting a set of pieces from a sheet of material in order to minimize a waste. The problem arises in various production processes, such as the glass, steel, wooden, paper or textile industries. The problem is of combinatorial nature and, thus, can be analyzed along the lines approporiate for this class of problems. The basis of such an analysis is always computational complexity. Following it, one may design an appropriate algorithm for solving the problem in question. Unfortunately, majority of cutting problems are strongly NP-hard, thus, unlikely to admit pseudopolynomial-time algorithms. Hence, they must be solved by approximation algorithms. One-dimensional and regular two-dimensional cutting problems allow for the application of approximation algorithms with a given accuracy (worst case behaviour). Unfortunately, no such method is known for irregular shapes, thus, heuristic approaches must be used. The above issues are presented in the following Sections.

* Technical University of Poznań, Institute of Computer Science, Poznań, Poland.
** Center for Agricultural Development, Poznań, Poland.

The organization of the paper is as follows. Section 2 contains problem formulation and a short introduction into the theory of computational complexity. Then, basic results for the one-dimensional version of the problem, are presented. In Section 3 two-dimensional regular (rectangular) problem is analyzed. A reference to several known algorithms is made here. Section 4 deals with irregular (not necessarily rectangular) case and several methods solving this problem are presented. Then, some hints for the use the two algoritms described in a decision support system are given.

2. BASIC CONCEPTS, DEFINITION AND RESULTS FOR ONE-DIMENSIONAL CASE

2.1. Problem formulation

One-dimensional cutting problems is the easiest version of the problem. It can be stated in the following way: given rods of unit length cut them into the set of elements a_i, $0 \leq a_i \leq 1$, $i = 1...n$, in order to minimize the number of rods used. This problem has the same nature as memory allocation or nonpreemptive task scheduling problems for computer systems. References to this problem can be found in [7, 10, 18]. This is the same as bin-packing problem.

Two-dimensional regular problem can be formulated as follows: given a set of rectangles with dimensions y_i and x_i, $i = 1...n$, distribute them into the minimal number of rectangular areas dimensioned Y and X. There are variants of this formulation. For example rectangular area to be filled with elements may have only one dimension limited while the other is to be minimized, rotation may be allowed or not, elements may appear once or more times. Reference to this problem may be found in [4, 5, 6, 7, 8, 9, 11, 13, 16, 17, 21, 26].

Two-dimensional irregülator problem definition differs from the above formulation in the fact that any shapes of elements are admitted. The problem has been discussed in [1, 2, 3, 15].

2.2 COMPUTATIONAL COMPLEXITY ISSUES

As was mentioned the complexity analysis is the basic for further studying problem. Thus, we will recall basic compexity definitions mainly with respect to decision problems, i.e. those requiring an answer of the "yes"–"no" type. Bin-packing (cutting) problem may be formulated in this way by asking a question if packing elements into the known number of bins is possible. On the other hand, plenty of optimization problems where some function is to be minimized (maximized), are known. Bin-packing in the original formulation is the optimization problem. There exists a close relation between decision and the optimization problems. If the optimization problem is easy to solve, then corresponding decision version is easy too. If decision version is difficult, then optimization problem is also difficult. We are going to use this relation further on. We consider only time complexity since space limitations are not of the great importance and may be avoided. Now we present basic definitions.

Decision problem Π is a set of parameters (sets, graphs, numbers) with values not necessarily asigned and a question with an "yes" or "no" answer. Assigning values to parameters creates **instance** I of problem Π. D_{Π} is a set of all instances. Data of I are encoded as a limited string $x(I)$ of symbols from known alphabet Σ according to some encoding rules. By an input size $N(I)$ we understand here the length of string $x(I)$. Only compact and precise encoding rules are allowed – redundant symbols are excluded, numbers are encoded with a base greater than 1. In practice $N(I)$ is assumed to be a number of the most important objects of the instance (tasks, polygnos, nodes in a graph).

Computational complexity of algorithm A solving problem Π one defines as a function $f_A(\mathrm{n}) = \max\{t$: t is a number of elementary computer steps needed to solve the problem for $I \in D_{\Pi}$ and $n = N(I)\}$.

Polynomial algorithm has computational complexity function (or complexity for short) $O(p(k))$ on deterministic Turing machine – DTM (or RAM model), where $p(k)$ is a polynomial, k is a size of the instance. Now we define classes of decision problems.

Class **P** consists of all problems solvable on DTM in polynomial time. (Hence, this class contains all problems solvable in polynomial time in practice).

Class **NP** consists of all problems solvable in polynomial time by nondeterministic Turing machine (NDTM). (In practice it is equivalent to the existence of a polynomial height branching tree in a branch and bound algorithm solving the problem). By the definition $\mathbf{P} \subseteq \mathbf{NP}$.

Polynomial transformation of problem Π_2 to Π_1 (we denote $\Pi_2 \, \lambda \, \Pi_1$) is the function f: $D_{\Pi 2} \rightarrow D_{\Pi 1}$, satisfying:

1. for every $I_2 \in D_{\Pi 2}$ answer is "yes" iff for $f(I_2)$ answer is "yes" too;
2. for every $I_2 \in D_{\Pi 2}$ time of computing f on DTM is bounded by polynomial in $N(I_2)$.

Decision problem Π_1 belongs to the class of **NP - complete** problems if $\Pi_1 \in \mathbf{NP}$ and for every $\Pi_2 \in \mathbf{NP}$, $\Pi_2 \lambda \Pi_1$. From the definition we conclude that if there is a polynomial algorithm for any NP - complete problem then any problem from NP may be solved by polynomial algorithm. This class contains such problems as 3 - dimensional matching, vertex cover, clique, hamiltonian cycle, set partition, graph coloring. Despite many trials, no polynomial algorithm solving any NP-complete problem is known. Thus, we expect these problems to be solvable only by exponential algorithms (and then P $\neq$ NP-complete class of problems).

On the other hand, certain NP-complete problems may be solved (quite efficiently, e.g. by dynamic programming) for the data appearing in the practice. Complexity of these algorithms is bounded by a polynomial of two variables – instance size $N(I)$ and maximum number value (appearing in the instance) max (I). We call them **pseudo-polynomial algorithm**. Such an algorithm may only be constructed for a **number decision problem** which does not have max (I) constrained by polynomial function of $N(I)$. we say that problem is **NP-complete in the strong sense** if it is in the class **NP** and there is polynomial p such that for D_Π limited to these instances only for which max $(I) \leq p(N(I)$, the problem remains NP-complete. From the above we see that no pseudo-polynomial algorithm is possible for the problem being NP-complete in the strong sense. To prove strong NP-completeness one applies strong **pseudo-polynomial transformation** (in which time bound for construction of function f is allowed to be pseudo-polynomial and some additional constraints on $N(I)$ and max(I) are imposed) and some known strongly NP-complete problem.

Now, let us consider again optimization problems. If a decision version of the problem is NP-complete, then an exact optimization algorithm for the original (optimization) version must be exponential. In such a case one applies polynomial **approximation algorithms** to obtain approximate solution. It is desired to know how far from the optimum is the solution generated by such an approximation algorithm, i.e. how precise it is.

For the approximation algorithm A and instance I we define ratio $S_A = \frac{A(I)}{OPT(I)}$ (for maximization problem), where $A(I)$ is the value of the objective function obtained by A and $OPT(I)$ is the optimal value.

Absolute performance ratio S_A for the algorithm A is

$$S_A = inf\{r \geq 1 : \forall_{I \in D_\Pi} S_A(I) \leq r\}.$$

Asymptotical performance ratio S_A^∞ is

$$S_A^\infty = inf\{r \geq 1 : \exists_{n \in Z^+} \forall_{I \in D_\pi, OPT(I) \geq n} S_A(I) \leq r\}$$

The closer S_A, S_A^∞ are to the 1 the better algorithm is.

For some combinatorial problems it can be proved that there is no hope of finding an approximation algorithm of certain accuracy (i.e. this question is as hard as finding a polynomial-time algorithm for any NP-complete problem).

Analysis of the worst case behaviour of an approximation algorithm may be complemented by an analysis of its mean behavoiur. This can be done in two ways. The first consists in assuming that the parameters of the instances of the considered problem Π are drawn from certain distribution D and then one analyzes the **mean performance** of algorithm **A**. One may distinguish between **absolute error** of an approximation algorithm, which is the difference between the approximate and optimal solution values and **relative error** which is the ratio the two. Asymptotic optimality results in stronger (absolute) sense is quite rare. On the other hand asymptotic optimality in the relative sense is often easier to establish [19, 22, 24].

It is rather obvious that the mean performance can be much better than the worst case behaviour, thus justifying the use of given approximation algorithm. A main obstacle is difficulty of proofs of the mean performance for realistic distribution functions. Thus, the second way of evaluating the mean behaviour of approximation algorithms, consisting of simulation studies, is stillused very often. In the later approach one compares solutions, in the sense of the values of a criterion, constructed by a given approximation algorithm and by optimization algorithm. This comparison should be made for a large representative sample of instances. There are some practical problems which follow from the above statement and they are discussed in [23].

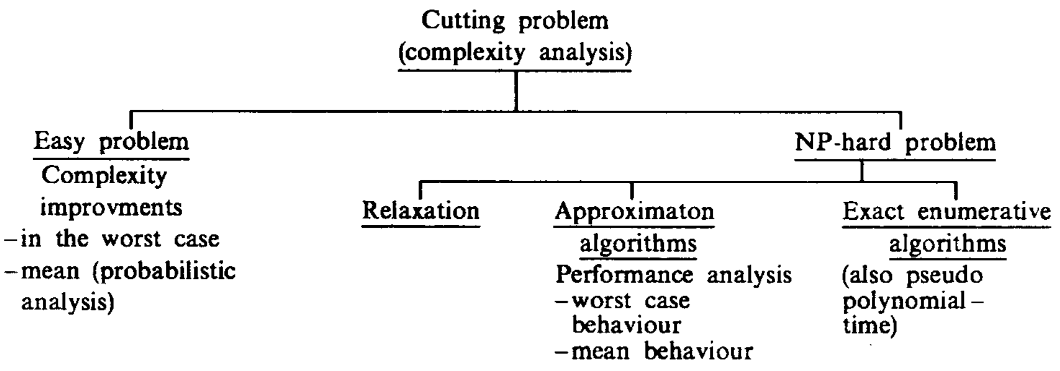

Fig.1. An analysis of cutting problem - schematic view

The third and last way of dealing with hard cutting problems is to use exact enumerative algorithms whose worst-case complexity function is exponential in the input length. However, sometimes, when the analyzed problem is not NP-hard in the strong sense, it is possible to solve it by a pseudo-polynomial optimization algorithm whose worst-case complexity function is bounded from above by the polynomial in the input length and in the maximum number appearing in the instance of problem. For reasonably small numbers such an algorithm may behave quite well in practice and it can be used in computers applications. On the other hand "pure" exponential algorithms have probably be exluded from application, but they may be used sometimes for other cutting problems which may be solved by off-line algorithms.

The above discussion is summarized in a schematic way in Fig. 1. Definitions from this Section are base for further analsis of our problem.

2.3. One-dimensional problem analysis

One-dimensional problem is the easiest version of the problem considered. From its analysis we can draw conclusions as to the general problem complexity.

One-dimensional cutting problem as stated in Section 2.1 is the same as bin-packing problem so we will refer here to the results for the latter. This problem is NP-complete in the strong sense for the decision version, this comes from pseudo-polynomial transformation of 3-partition problem [12]. 3-partition problem is:

Parameters: limit $B \in Z^+_{3q}$, set A, $|A| = 3q$, $q \in Z^+$, value $s(a_i) \in Z^+$ for every $a_i \in A$, $B/4 < s(a_i) < B/2$, $\sum_{i=1} s(a_i) = Bq$.

Question: does there exist a partition of A into q disjoint subsets $s_1, s_2, \ldots, s_q$ satisfying $\sum_{a_j \in s_i} s(a_j) = B$ for $i = 1, \ldots, q$?

Proof is easy we see that 3-partition is a special case of bin-packing problem.

Now we know that the problem will not be solved by a polynomial algorithm (if $\mathbf{P} \neq \mathbf{NP}$), yet for the fixed number of element sizes there exists linear time solution [7].

Assume p is integer such that sizes of elements are from the set $\{1/p, 2/p, \ldots, (p-1)/p, 1\}$ and we pack them into unit size box. Elementary instance E is a set of elements satysfying $\sum_{i=1}^{n} s(a_i) \leq 1$. Data of the instance may be written as a p-dimensional vector $\bar{v} = [v_1, \ldots, v_p]$ where v_i is a number of elements of size i/p. Thus every solution is a set of elementary instaces and the problem can be stated as a partition of a set of elements into the minimal number of elementary instances. We see that number K of elementary instances is fixed. We will denote them as p-dimensional vectors $\bar{b}_1, \bar{b}_2, \ldots, \bar{b}_K$ called **elementary vectors.** Now our problem can be formulated as an integer linear programming: find $\alpha_1, \alpha_2, \ldots, \alpha_k$ that minimize $\sum_{i=1}^{K} \alpha_i$, subject to $\sum_{i=1}^{K} \alpha_i \bar{b}_i = \bar{v}$ and $a_i \geq 0$.

A general version of integer linear programming is strongly NP-complete, but for a fixed number of variables K it can be solved in polynomial time [20]. Using the above transformation of the input data one may solve the problem in question in linear time. This is rather a theoretical result since a number of variables for practical situations may be great. Then complexity function though linear in the number of elements has a large constant before it. This constant grows exponentialy with K.

There exists a number of approximation algorithms for bin-packing problem (thus for one-dimensional cutting). We are going to mention only most important. **First fit (FF)** algorithm – assigns element to the box with the lowest possible number. **Best fit (FF)** algorithm – assigns element to the box with the minimum remaining capacity.

Let S_{FF}, S_{BF} denote the absolute performance ratio for the FF and BF algorithms respectively and C^* – a number, of boxes used by an optimal solution. Then it can be shown [25] that

$$S_{FF} = S_{BF} = \frac{17}{10} + \frac{2}{C^*}.$$

First fit decreasing (FFD) algoritm is a FF algoritm with elements assigned in nonincreasing order of their sizes. **Best fit decreasing (BFD)** algorithm is a BF algorithm for elements scheduled in nonincreasing order of their sizes.

From [14, 18] the asymptotic performance ratios for FFD and BFD are known (here sizes of the elements a_i are drawn from the interval $[0, \alpha]$):

$$S^{\infty}_{FFD}(\alpha) = \begin{cases} \frac{11}{9} & \text{for} \quad \alpha \in (\frac{1}{2}, 1] \\ \frac{71}{60} & \text{for} \quad \alpha \in (\frac{8}{29}, \frac{1}{2}] \\ \frac{7}{6} & \text{for} \quad \alpha \in (\frac{1}{4}, \frac{8}{29}], \\ \frac{23}{20} & \text{for} \quad \alpha \in (\frac{1}{5}, \frac{1}{4}] \end{cases}$$

$$S^{\infty}_{BFD}(\alpha) = \begin{cases} \frac{11}{9} & \text{for} \quad \alpha \in (\frac{1}{2}, 1] \\ \frac{71}{60} & \text{for} \quad \alpha \in (\frac{8}{29}, \frac{1}{2}] \\ \frac{7}{6} & \text{for} \quad \alpha \in (\frac{1}{4}, \frac{8}{29}], \\ 1 + \frac{k-2}{(k-1)k} & \text{for} \quad \alpha \le \frac{1}{4} \quad k = \frac{1}{\alpha} \end{cases}$$

(The last line of $S^{\infty}_{BFD}(\alpha)$ is a proposition only). Some other approximation algorithms are surveyed e.g. in [10]. In this Section we have shown that one-dimensional cutting problem in general case is NP-complete in the strong sense and we should not expect polynomial algorithms. Best approximation generate solutions worse about 20% than optimum in the worst case, in practice an average difference is less than 10%.

3. TWO-DIMENSIONAL REGULAR CUTTING PROBLEM

3.1. Introduction

The problem of two-dimensional regular cutting defined in 2.1 has several variants. For all the cases the common assumptions are
a) pieces can not overlap each other or the edges of material
b) pieces can not be inverted (as in the mirror).

For some cases a raw material may consist of rectangular sheets of material then the objetive is to minimize its number. Sometimes the ribbon of material is given then one has to minimize a length while a width is constant. On the other hand, if the area of the material is one rectangle then the aim is to pack elements that minimize a weste. Rotations of elements are rather not considered and if any then 90 degrees rotation are assumed. In some cases only guillotine cuts are allowed, i.e. from edge to edge parallel to the other pair of edges. We know that one-dimensional version has been already NP-complete in the strong sense. Thus in such a situation one can construct exponential and optimal algorithms or polynomial approximation ones. In the following subsections we describe two optimal algorithms and several approximatin ones adjusted to the different versions of the problem.

3.2. Iterative combinatorial algorithms

Christofiedes and Whitelock's branch and bound algorithms [8]. This algorithm solves a single sheet problem and it is based on a tree – search procedure. It limits the number of nodes imposing necessary conditions on the optimality of patterns to be cut. This is done by means of transportation routine and dynamic programming routine.

Assume $A_0 = (L_0, W_0)$ is a sheet of material with dimensions L_0 (length) and W_0 (width). R is a set of rectangles $R = \{(a_1, b_1), \ldots, (a_m, b_m)\}$. Every rectangle has value v_1 and maximal number of appearances in the resulting pattern l_i. Every number in the problem is integer, cuts are of a guillotine type, and rotations are not allowed. The problem is to maximize $z = \sum_{i=1}^{m} \zeta_i v_i$, subject to $0 \leq \zeta_i \leq l_i$, $i = 1, \ldots, m$, $\zeta_i \in Z^+$ and there exists a sequence of cuts of A_0 resulting in ζ_i rectangles of the i'th type.

The algorithm has two steps – generating the tree of all possible cuts and scanning it for the best solution. Every node of the tree represents a possible cut. During the generation phase symmetrical cuts are exluded. For example cuting of rectangle (p, q) in the point e is symmetrical with the cutting in the point p-e. Such a symmetries are exluded by analyzing in the rectangle (p, q) only points with $x \leq \lfloor p/2 \rfloor$ and $y \leq \lfloor p/2 \rfloor$ (where $\lfloor a \rfloor$ is the greatest integer not greater than a). Repetitous cuts are eliminated by imposing succession of cuts – for example if we cut at point $x = a$ then every succeeding cut has to be done at $x \geq a$. Only **normalized cuts** are considered (cf. fig. 2) that is in points which are linear

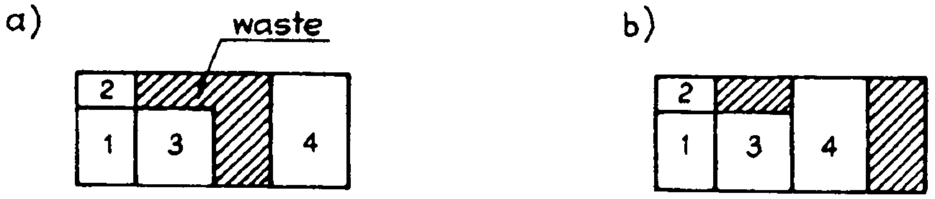

Fig.2. a) not normalized cut, b) normalized cut

combinations of sizes of elements. This exlude cutting with waste inside a pattern.

Possible cuts of rectangle (p, q) resulting in the elements of set R are entries of set S^q for cutting in Ydirection and T^p for X direction. Now we describe how to find S^q, T^p is found in the same way. We use function $f_m(x)$ to generate S^q. This function can be computed recursively as follow (rectangles are ordered according to nonincreasing value of b_i): for $i = 1,\ldots,m,\ x = 0,\ldots,L_0$,

$$f_i(x) = min\{f_{i-1}(x), \max_j\{b_i, f_{i-1}(x - ja_i)\}\}\ j = 1\ldots min(l_i, \lfloor x/a_i \rfloor) \text{ if } x \geq a_i$$

$$f_i(x) = f_{i-1}(x) \text{ if } x < a_i,\ f_0(x) = \infty.$$

State of a node in the tree is described by the list $L$of rectangles cut on the path from the root. rectangle is represented on that list by vector (p, q, x, y), (p, q) being sizes of a current rectangle and x, y are describing the following cuts if the rectangle is chosen. In order to find an optimal solution, for every node, an upper bound estimation of the objective function is computed. This estimation is computed in two ways. Suppose $H_0 \subseteq L$ is a list of rectangles that will not be cut any more. Estimation z may be computed as a total flow in a special transportation problem that assigns elements from the set R to H_0. Upper bound estimation for nodes with still possible cuts can be computed by dynamic programming procedure for relaxed version of the problem – not considering limits l_i [13]. If computed value z^* is better than previously found $\underline{z}$, then one substitutes previous solution with the current one and the algorithm proceeds to the next node.

Wang's combinatorial algorithm [26,21]. This algorithm is a combinatorial one that generates guillotine cutting patterns by successive adding pieces of groups pieces to each other. These cut patterns are normalized in the sense of the previous algorithm. To avoid explosive growth of number of partial solutions the algorithm rejects solutions with a waste exceeding some percentage of stock sheet area or for the second version with a waste exceeding a percentage of the area of a partial solution.

Let us denote by S_k a partial solution generated at iteration k, by F_k – a list of all partial solutions generated during iteration k, by L_k – a list of all partial solutions generated until iteration k and by β – rejection parameter $0 \leq \beta \leq 1$. Wang's algorithm can be formulated as follows

```
choose $\beta$:
$L_0, F_0 := R$;
$k := 0$;
while $F_k$ not empty do
    $k := k + 1$;
    $F_k := \{\}$;
    generate all partial solutions $S_k$ adding elements of $F_{k-1}$, to all
    elements of $L_{k-1}$;
    for each $S_k$ do
        if $S_k$ fits in the stock sheet
            and the number element $i$ appears in $S_k$ is not greater than $l_i$
            and the waste in $S_k$ is not greater than $\beta L_0 W_0$
            then $F_k := F_k \cup S_k$;
    $L_k := L_{k-1} \cup F_k$;
$M = k$;
choose the element of $L_M$ with the least total waste.
```

It is shown in [26] that if the waste of the best pattern is not greater than $\beta L_0 W_0$ then this pattern is optimal (there is no pattern with a smaller waste). A modification of the above algorithm (described in [21]) is done by means of dynamic programming algorithm for unconstrained number of elements [14] and it improves the way expected waste for partial solution is computed. Thus, worse solution are rejected earlier.

3.3. Approximation algorithms

There are many approximation algorithms. We describe only some, which, in our opinion, are the most important. If not stated otherwise the unit width of the stock sheet is assumed, a length is to be minimized, and rotations are not allowed. For a given list L of rectangles an approximation algorithm generates solution with stock sheet length $A(L)$ while optimum is $\mathrm{OPT}(L)$. We use the absolute performance ratio

$$A(L) \leq \lambda\, \mathrm{OPT}(L)$$

and an asymptotic one

$$A(L) \leq \mathrm{OPT}(L) + \beta.$$

Let us pass now to the algorithms.

Bottom left decreasing (BLD) algorithm. Rectangles given on the list $L$are ordered according to nonincreasing sizes. Put the next element from Las low and as much to the left as possible. For every L: BLD $(L) \leq 2\,\mathrm{OPT}(L)$, thus algorithm BLD generates worst case solution 100% worse than optimal one.

The following two algorithms [11] are so called **level oriented algorithms**. The level-oriented name comes from the fact that pieces are located in layers. The first layer bottom is a bottom of the stock sheet, the following are marked by the top of the first (that is highest) element in the preceding layer. Elements on L are ordered according to nonincreasing height.

Next fit decreasing height algorithm (NFDH) -- if there is not enough room at the current (top) level to place a rectangle considered, then create a new level (Fig. 3).

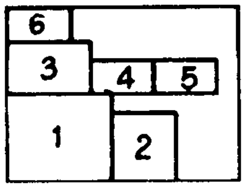

Fig.3. An example solution by NFDH algorithm

First fit decreasing height algorithm (FFDH) – puts rectangles at the lowest possible level and if it is not possible creates new one (Fig. 4).

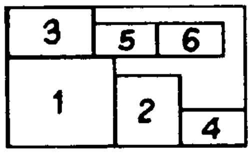

Fig.4. An example solution by FFDH algorithm

Asymptotic performance ratio for NFDH is

$$\mathrm{NFDH}(L) \leq 2\mathrm{OPT}(L) + 1,$$

for FFDH

$$\mathrm{FFDH}(L) \leq 1.7\,\mathrm{OPT}(L) + 7.3$$

and for sizes of elements not exceeding λ

$\mathrm{FFDH}(L) \le (1 + 1/m)\mathrm{OPT}(L) + (2 + 1/m)$, where $m = \lfloor 1/\lambda \rfloor$,
for squares

$$\mathrm{FFDH}(L) \le \frac{3}{2}\mathrm{OPT}(L) + 2.$$

Spilt fit algoritmh (BF) [11]. Let $m \ge 1$ be the greatest integer such that all rectangles have widths not greater than $1/m$. The list of pieces is ordered according to the nonincreasing heights. Spilt L into two lists L_1, L_2. L_1 consists of elements of widths greater than $1/(m+1)$, L_2 contains the remaining elements. First, put L_1 rectangles with FFDH algorithm then move the layers wider than $(m+1)/(m+2)$ to the bottom of the pattern down under layers thinner than $(m+1)/(m+2)$. Thus, there is a free rectangular area $1/(m+2)$ wide. Put into this area L_2 elements using FFDH algorithm. Place remaning L_2 rectangles above the pattern for L_1 (Fig. 5).

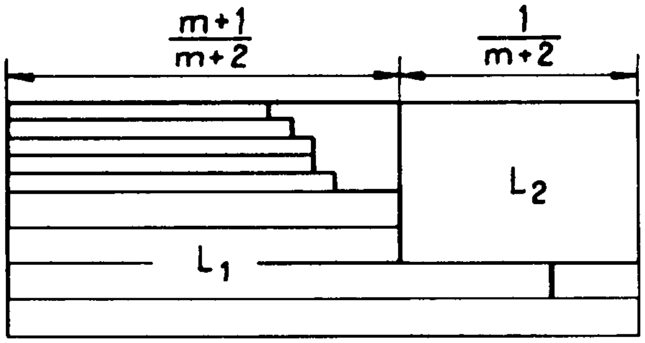

Fig.5 SF algorithm layout

Asymptotical performance ratio for SF algorithm is

$$\mathrm{SF}(L) \le \frac{m+2}{m+1}\mathrm{OPT}(L) + 5.$$

Up down algorithm (UD) [4]. this algorithm is equivalent to NFDH algorithm for rectangles thinner than 1/5; for the wider several strategies are mixed. This algorithm is a bit more sophisticated than previously mentioned and we will only give its brief outline. The algorithm splits the stock into the five regions numbered from the bottom of the stock to its upper part. In the regions $1 \le i \le 4$ rectangles being wide $1/(i+1)$ trough $1/i$ are packed according to BL (bottom left) algorithm. Thus, there remains some free area in the right top corner. Thus more rectangles can be placed in the column from the top down. When all elements wider than 1/5 are placed in regions

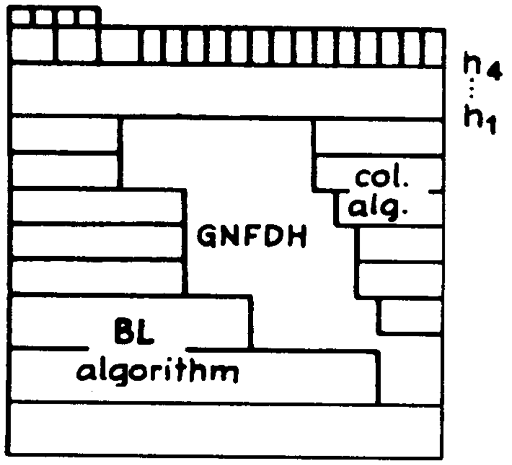

Fig.6. UD algorithm allocation layout

$1 \leq i \leq 4$, then the remaining rectangles are put into the slot between elements located by BL and column algorithm (cf. Fig. 6). This is done by means of generalized next fit decreasing algorithm (GNFDH). Asymptotic worst case behaviur for rectangles not exeeding height H is

$$\mathrm{UD}(L) \leq \frac{5}{4}\mathrm{OPT}(L) + \frac{53}{8}H.$$

Algorithms described above have to work "off line" since it is necessary to know the set of rectangles before the start, and more over these pieces have to be sorted. For certain applications however this is not possible to wait until all parts are known to sort them. For example we can not wait for the arrival of all parts to computer their allocations in the warehouse area. There are certain so-called **shelf algorithms** predestined to work "on line" without initial sorting of elements or even knowing them. These algorithms are modifications of NFDH and FFDH algorithms. additional free space is created to handle the elements of bigger size expected to come later. The parameter r is a measure of that additional space. Every created shelf has value r^k (for some k), and an element of height h, $r^{k+1} \leq h \leq r^k$ has to be packed into the shelf of r^k height.

Next fit shelf algorithm with parameter r (NFS_r) – puts rectangle as far to the left on the highest (last) shelf as possible, and if this is not possible, a new shelf is created.

First fit shelf algorithm with parameter r (FFS_r) – puts rectangle at the lowest possible shelf as to the left as possible, and if this is not possible, a new shelf is created.

It can be shown [6] that for $0 < r < 1$ and rectangles not higher than H

$$\mathrm{NFS}_r = \frac{2}{r}\mathrm{OPT}(L) + \frac{H}{r(r-1)},$$

$$\mathrm{FFS}_r = \frac{1.7}{r}\mathrm{OPT}(L) + \frac{H}{r(r-1)}.$$

For the case with multiple stock sheets of the same limited sizess the objective is to minimize a number of sheets used. There exists [9] HFF **algorithm** for this purpose. HFF is a mixture of FFDH and FFD. First, according to FFDH a patern with levels in the unlimited height stock is constructed, then levels are assigned to the stock sheets according to FFD algorithm. Asymptotic performance ratio for HFF is

$$\mathrm{HFF}(L) \le \frac{17}{8}\mathrm{OPT}(L) + 5.$$

In this chapter a very short insight into the group of the algorithms dealing with two-dimensional regular cutting has been presented. There are two main groups of algorithms – optimal exponential combinatorial ones and those based on approximation approaches with the worst case bounds known. Due to the progress in the computer hardware speed the sizes of problems that can be solve by optimal algorithms are growing [21]. On the other hand average behaviour for realistic cases of approximation algoritmhs is much better than the worst case estimates suggest.

4. IRREGULAR TWO-DIMENSIONAL CUTTING PROBLEM

4.1. Introduction

This problem admits any shapes of elements. Strong NP-completeness of decision version of the problem implies the lack of the polynomial optimization algorithm. Worse still, as far as we know, there are neither algorithms with known worst case behaviour bounds nor algorithms computing optimal solution in any way. In practice, only experimental evaluation and comparison on the base of some objective function (waste, time), is possible. The only method optimal in some sense has been proposed by Adamowicz [1]. This metod involves iterative solution of an integer programming problem followed by an adjusting procedure, which generate new constraints for the next iteration until an optimal solution is constructed. However, this approach is so complex that experimental program is either not completely usable or implements very simplified version of the method.

The other methods known for the problem in question are heuristics using different approaches to the problem. These algorithms though polynomial and approximate consume a lot of time involving hard numerical computations. From this fact we can draw conclusions: there is a trade-off between the solution time and quality of the solution. In this context the importance of hybrid – semiautomatic methods increases, where tentative solution is automatically generated and the interactive improvements are allowed by conversational display unit.

We outline below ideas of four methods: by Adamowicz and three heuristics [2, 3, 15]. The first and the second heuristics have been implemented in the program described in Section 5.

4.2. Algorithm by Albano-Sapuppo

This algorithm [2] is based on the search method for optimal solution in the directed graph of all partial solutions using several heuristic techniques that increase the search power. Pieces are assumed to be irregular polygons without holes, the sheet is a rectangle. Discretized step rotations are allowed. The goal is to minimize the waste or (better) the length of produced packing.

Many problems in artificial intelligence and operations research are solved by a technique based on searching through a "space" of candidate solutions. The above approach utilizes this technique. The set of states reachable from the initial state can be seen as a directed graph with nodes– states of allocation and arcs – allocation operations. The solution is a search process for a path from the initial state to the member of the set of final nodes. Search process can be organized in the following way:

1. Put the start node on the list GENERATED.
2. if GENERATED is empty exit with failure.
3. Select a node from GENERATED according to some rule **R** and put it on a list EXPANDED, call it **n**.
4. If **n** is a final node exit with a solution path.
5. Expand **n**, that is generate all its successors. If there are no successor go to 2, otherwise put them on GENERATED and go to 2.

Usually rule **R** selects a node with the smallest **evaluation function** which is a sum of an estimate of the cost of the path from the starting to the current one and estimate of the cost of the path from the current node to the final one.

Let us consider A_0 – an initial allocation containing no elements. A_i is a final allocation if there are no more elements to allocate. **Added waste** for the allocation A_i of piece p_i is defined as follows

$$\text{added\_waste}(A_i) = \text{space}(A_{i-1}) - (\text{space}(A_i) + \text{area\_of}(p_i)),$$

where space (A_i) is the area on the right side of the profile (rightmost borders of rightmost pieces), i.e. the area which may be used to allocate piece p_i (Fig. 7).

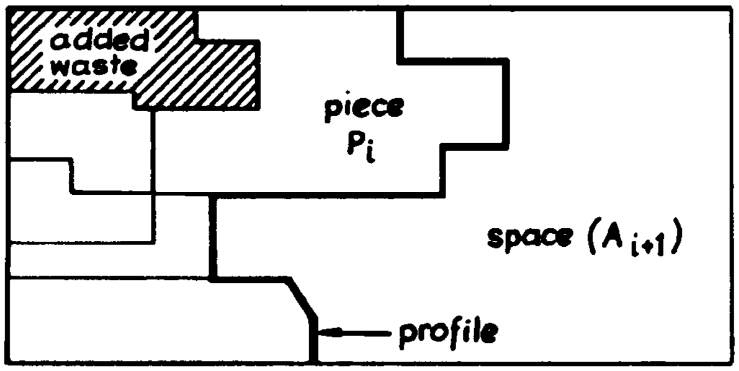

Fig.7. Partial solution pattern

The wasteA_i is recursively defined as waste$(A_0) = 0$,

$$\text{waste}(A_i) = \begin{cases} \text{waste}(A_{i-1}) + \text{added\_waste}(A_i) & \text{if } A_i \text{ is not final} \\ w*l - \sum \text{area\_of}(p_i) & \text{otherwise} \end{cases}$$

In order to transform the optimal allocation problem into the search of an optimal path in a state space, an initial state corresponds to A_0, the cost of an arc from S_i to S_{i+1} is the added waste produced by allocation A_{i+1}. The procedure to implement the heuristic search method can be stated as follow:

```
begin
initial conditions;
input pieces and stock sheet descriptions;
let the CURRENT_NODE be the initial state;
while CURRENT_NODE $\neq$ final node do
for all pieces left to be allocated do
    for all orientations do
        apply placement policy; (waste computation)
        apply evaluation function and
        append successor to the list GENERATED
    end
end;
set the CURRENT_NODE EXPANDED;
let the CURRENT_NODE be the "best" successor in GENERATED;
end;
plot solution;
end
```

There are some techniques to increase heuristic search power because the above procedure can not be applied to any realistic applications without rejecting "bad" nodes.

1. Evaluation function – the problem is how to evaluate cost from the current to final node. It should always be lower than the waste that would result in the optimal solution if the piece in question were to be included. A possibility is 0, but it has been prefered to drop the admissible property because the optimality of the solution is not critical and "good"will be enough. Thus the authors suggest a constant percentage of unplaced elements as an estimation of the cost.

2. Successor limitation – since the step of evaluation for all pieces and orientations is one consuming the most of the time, several limitations have been added.

a) From unplaced pieces only the one which produces the leftmost lowest allocation will be considered in step b).

b) Only the fixed number of leftmost allocated pieces are preserved for step c).

c) For allocation from b) only the fixed number of successors will be generated.

d) When the list of generated nodes becomes full, the tree is pruned by erasing the node with the highest evaluation functions.

3. Evaluation function discretization – continuation of the search along paths with small differences is allowed only within some precision.

4. Expansion band – when the search is at the *k-th* level, the next node to be expanded has to be at level at most *k-t* where *t* is given threshold.

5. Termination condition – at the end of the routine after the first final node is found the procedure develops all the posible search trees within the expansion the band and the final solution will be th best one.

6. Profile simplifications – at each step the profile is simplified in order to exlude all the areas on the left side of the vertical line through the leftmost point of the last allocation pieces.

There are several notions related to this algorithm important especially in finding piece allocations.

No-fit-polynon (NFP) for a pair *A*,*B* of pieces – completely describes all those positions where the reference point of *B* may be placed in order to have *B* touching *A* without overlapping (Fig. 8).

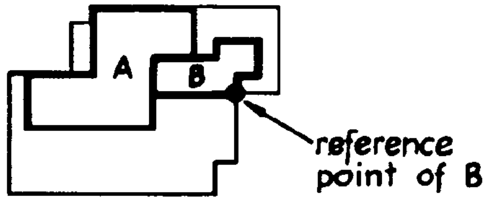

Fig.8. NFP for *A* and *B*

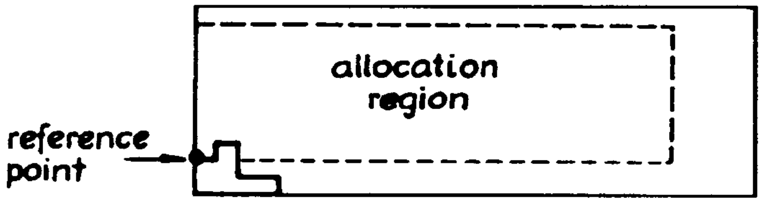

Fig.9. Allocation region

Allocation region for a given resource and piece – the area in which the reference point of the piece can validly fall (Fig. 9).

This algorithm has been tested thorughly and its performance and reliability indicates that it favourably compares with some others.

4.3. Algorithm by Art

This algorithm [3] is very similar to the Albano-Sapuppo algorithm in the way of handling geometrical entities. It was the base to create previous algorithm. Here, more stress is put to geometrical problems while before mainly heuristic search organization has been considered. This Section may be both description of independent algorithm and supplement for Albano-Sapupppo algorithm. There are the following assumptions made

1. There is distinguished piece direction (orientation) parallel to the distinguished direction of the material let us say X axis. Thus rotation are not allowed.
2. Flipping of pieces is not allowed (mirror symmetry).
3. The stock sheet is a rectangle with given width, and its length is to be minimized.

Every piece is defined by the sequence of line sections approximating its edge with a precision required. We exlude here any curve edges because of the growth of computational complexity. Distinguished element direction is parallel to X axis. There is a distinguished point in the sequence of vertices called reference point and its location defines uniquely the element position. Both concave and convex polygnos are admitted but because of higher computational complexity concave objects are exluded and approximated to the convex equivalent.

Important notion is the **envelope** – it is a sequence of line sections defined for every piece type. This is a trace of reference points created during moving element around allocation region in contact with its border. At the start of algorithm envelope is a rectangle but succesively after alocation of any element it is modified with the no-fit-polygnos (NFP – Section 4.2) (Fig. 10). Envelope can be understood as a border of an allocation region.

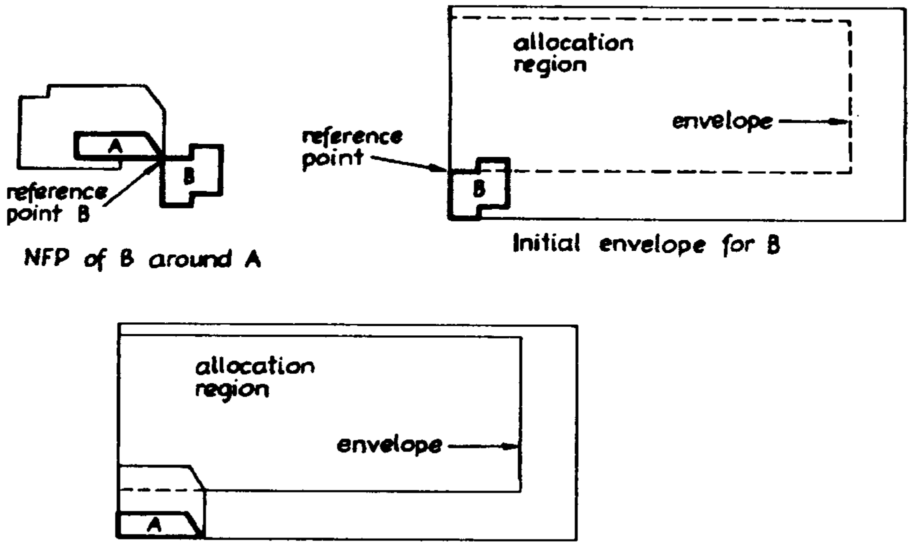

Fig.10. Modyfication of element *B* envelope after allocation of element *A*

Process of allocation is deterministic and sequential. Reference point defining element location is to be placed on the envelope – this guarantees contact with other elements without overlap. In order to minimize total waste, elements are allocated from the left side to the right according to the following heuristic rules:

1. Place element with the minimal envelope *X* dimension value first.
2. Place element in a certain place with minimal added, waste.
3. Split big groups of elements into the smaller one or into the separate elements to put in certain places separately.
4. Place element with greater area first.
5. Place element minimizing stock sheet length first.

Position on this list do not reflect importance of rules, it should be rather experimentially adjusted to typical problem instances.

4.4. Optimal algorithm by Adamowicz

This algorithm [1] allows the most general version of the problem to be solved. Various elements and stock sheets as well as multiple linear, logical and geometrical constraints associated with them are considered. Solution is obtained by iterative application of a two-stage procedure. The first is a linear programming problem; its solution minimize linear objective function subject to linear constraints. The second, geometrical, stage checks if the set of elements can be allocated feasibly satysfying geometrical constraints. If the solution does not exist, then new linear constraints resulting from the information obtained in the second step are enerated for the new iteration.

This method allows elements and stock to be holed, irregular and even not continuous. There are geometrical constraints of the two types: absolute – that bound locations on the stock area and relative – defined in relation to the other elements. Logical and linear constraints are for example number of elements of certain type required, a maximal number of types of elements, a ratio of quantities in which two types of elements are cut and others. The algorithm can be stated as an iterative application of the three following procedures.

1. A solution of a linear programming problem with rejection of redundant constraints.

2. Checking if the linear programming solution satisfies geometrical constraints and if not saving an information about feasible allocations.

3. Creating new linear constraints, including information gained in step 2.

Elements and material area are defined by sequence of vertices on its polygonal border. If the element is not compact (i.e. includes holes) it has to be defined as a group of compact elements with an additional relative location constraint. A location of an element is given by parameters: x, y and rotation angle from an initial orientation. During the processing phase of geometrical constraint an evelope (see section 4.3) for absolute constraints and NFP (Section 4.2) for relative constraints, are computed.

Linear programming phase gets into the consideration:

- linear constraints of the number of types of elements;
- logical constraints on the presence of types of elementes, this can be converted into linear constraints as well;
- geometrical consraints on the relative location of certain types of elements – since this is inherently nonlinear mainly geometrical phase of algorithm deals with kind of constraints;
- linear objective function maximizing income or number of elements allocated, density of allocation and/or minimizing the cost, waste etc.

As a result of linear programming phase solution one gets a set of elements to be allocated. Geometrical phase checks if allocation of elements chosen in the previous phase is possible. The elements are considered according to their nonincreasing areas. This procedure searches in the space of all possible locations of elements maximizing number of elements allocated.

As we can see this algorithm is very complex and rather difficult to handle in practice. Experimental programs are either completely inoperable or are simplified versions of the method. Let us note that the computational complexity of the geometrical phase grows exponentialy in the number of objects and orientations due to the search of candidate allocations, thus this method is rather a theoretical one.

4.5. Algorithm by Gurel

Every element is represented in this method [15] as a node in the graph. There is an arc between nodes if the corresponding elements are touching each other (or in other words are in contact). Rectangular area of the stock is represented as a disk called **marker disk**. Nodes corresponding to pieces in contact with the stock border are on the circle of marker disk, while nodes inside it correspond to elements without common points with edges of the raw material. In this way, a graph reflecting elements allocation inside the marker disk, has been created. There are vertical paths (strings of nodes) inside marker disk with at least one node on the border of that disk. The first string of that type corresponds to pieces in contact with the left border of material; we denote it **boundary break** – **BB1** for short. The second string of that type consists of nodes in contact with the right border of the stock area and will be denoted **BB2**. All other vertical strings of nodes between BB1 and BB2 we name **intermediate breaks (IBn** – for short, **n** being **IB** number). Interesting feature of the solution is that in the final layout there must be at least one horizontal string of pieces forming horizontal break. This will be called **cobreak**. Cobreaks as well as breaks may either lay along the boundary of marker circle or cross the marker disk by joining BB1 and BB2. Therefore there are boundary cobreaks or intermediate cobreaks. During the implementation of the method it has been observed that the biggest waste is created at the (right side) and of the layout. Therefore IBs are allocated according to minimal waste from both ends of the layout to the center of area. Thus, the method by Gurel can be formulated as follow

1. Initial computations.
2. Create boundary break BB1 and them BB2.
3. While not allocated elemens exist, create an internal break IB and move it to the right or left group of breaks.
4. Join left and right groups of breaks.

The algorithm requires some additional parameters a, b, c, s. Coefficients a,b,c are used create four groups of elements relative to their areas. Let us denote by P_{max} area of the largest element and by P an area of element considered. There are the folowing groups of pieces (depending on the areas):

$L1$: $aP_{max} \leq P$,

$L2$: $bP_{max} \leq P < aP_{max}$,

M: $cP_{max} \leq P < bP_{max}$,

S: $P < cP_{max}$.

Elements from $L1$ are prefered in BB1, from $L2$ and $L1$ – in BB2, M – in IBs. Elements from S are not allocated by this algorithm and should be placed interactively by an operator. This follows an observation that big differences of elements sizes reduce quality of the solution. In practical cases a, b, c are set to

a:40 ÷ 85%, b:25 ÷ 60%, c:0 ÷ 25%. The bigger the differences in area sizes are, the greater a, b, c should be. This method admits rotations of elements with a given step. Now, in the short outline we describe some procedures of the algorithm.

During the initial phase, for every element and every orientation the following parameters are computed:

- area,
- reference point,
- coordinates of boundary points $BP_1, \ldots, BP_8$ (Fig. 11),
- waste areas $\delta_1, \ldots, \delta_8$,
- waste $\tau_{A1}, \ldots, \tau_{C4}$ for appropriate sweeping directions

$$\tau_{A1} = \delta_7 + \delta_8 + \delta_1 + \delta_2 + \delta_3, \quad \tau_{A2} = \delta_3 + \delta_4 + \delta_5 + \delta_6 + \delta_7,$$
$$\tau_{B1} = \delta_5 + \delta_6 + \delta_7 + \delta_8 + \delta_1, \quad \tau_{B2} = \delta_1 + \delta_2 + \delta_3 + \delta_4 + \delta_5,$$
$$\tau_{C1} = \delta_7 + \delta_8 + \delta_1, \quad \tau_{C2} = \delta_3 + \delta_4 + \delta_5,$$
$$\tau_{C3} = \delta_1 + \delta_2 + \delta_3, \quad \tau_{C4} = \delta_5 + \delta_6 + \delta_7.$$

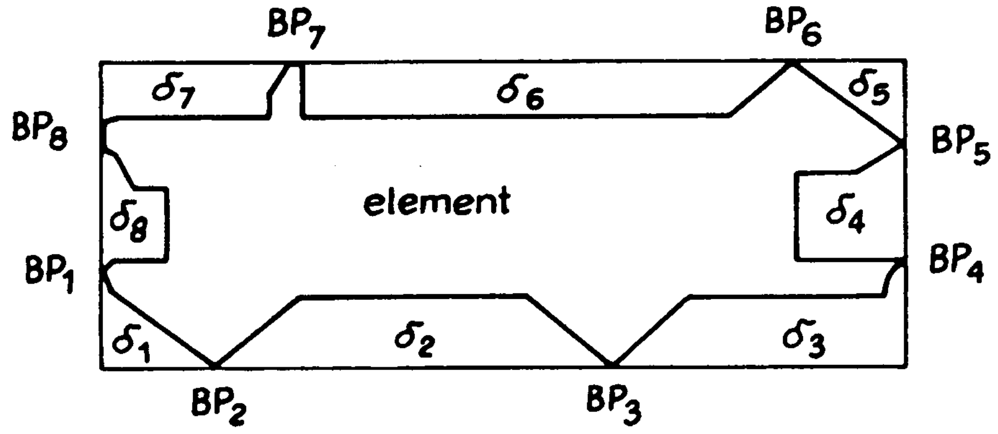

Fig.11. Object's boundary points and waste areas in Gurel's algorithm

During the initial phase elemens are assigned to groups L_1, L_2, M, S.

In order to create BB1 and BB2 we choose groups of elements minimizing waste between the borders of stock and element. Bigger pieces from BB1, BB2 are considered first. In order to check if the element fits into the BB1 or BB2 we evaluate how deeply the element in question may coincide with elements previously allocated in the string and compare the result with the width of element and width of a "slot" in the string. Internal breaks (IBs) are created a little bit easier only with comparisons of wastes τ_{C3}, τ_{C4}. The components of IB are moved to each other in the vertical direction. In order to minimize the waste indside IB, pieces are moved horizontally within some band with a given step s.

This method seems to be faster than previously described due to its graph theoretic approach and good heuristic methods to carry on computations. This is done with a little reduction of effieciency in area usage. There are several factors influencing efficiency of algorithm, for instance number of elements, number of types of elements, parameters *a, b, c, s,* width of the stock sheet etc. This method seems to fit well into semiautomatic approach requirements.

5. CONCLUSIONS

In the paper, after preliminary complexity investigations, we have described four methods for irregular two-dimensional cutting. The first three of them are a bit similar in the geometrical notions used. We think that Adamowicz approach though difficult to handle in practice, shows interesting directions to create more general systems for cutting problem. The three remaining methods are applicable in practice and broadly described in references. In the following paper a decision suport system, using the two of the described methods, will be described.

REFERENCES

[1] Adamowicz M., *The Optimal Two-Dimensional Allocation of Irregular Multiply-Connected Shapes with Linear Logical and Geometric Constraints,* N.Y. Uniw. of Tech. Report 403-9, New York 1969.

[2] Albano A., Sapuppo G., Optimal Allocation of Two-Dimensional Shapes Using Heuristic Search Methods, *IEEE Trans. an Systems, Man and Cybernetics,* vol. SMC-10, no. 5, May 1980.

[3] Art R.C.Jr., *An Apprach to the Two-Dimensional Irregular Cutting Astock Problem,* IBM Report 20-2006, Cambridge 1966.

[4] Baker B.S., Brown D.S., Katseff H.P., $\frac{5}{4}$ Algorithm for Two-Dimensional Packing, *J. of Algorithms,* 2 (1981), 348-368.

[5] Baker B.S., Coffman E.G.Jr., Rivest R.L., Orthogonal Packings in Two Dimensions, *SIAM J. Comput.* 9 (1980), 846-855.

[6] Baker B.S., Schwartz G.S., Shelf Algorithms for Two-Dimensional Packing Problems, *SIAM J. Comput.,* 12 (1983), 508-525.

[7] Błażewicz J., Ecker K. A Linear time Algorithm for Restricted Bin-Packing and Scheduling Problems, *Operations Research Letters* 2, no. 2, 1983, 80-83.

[8] Christofides N., Whitelock C., An algorithm for two-dimensional Cutting Problems, *Operations Research* 25, 1977, 30-44.

[9] Chung F.R.K., Garey M.N., Johnson D.S., On Packing Two-Dimensional Bins, *SIAM J. Alg. Dis. Math.* 3 (1982) no. 1, pp.66-76.

[10] Coffman E.G.Jr., Garey M.R., Johnson D.S., Approximation Algorithms for Bin-Packing – an updated survey in G. Ausiello, M. Lucertin, P. Serafini (eds.), *Algorithms for Computer Systems Design*, Springer Verlag, Wien 1984, 49-106.

[11] Coffman E.G.Jr., Garey M.R., Johnson D.S., Tarajan R.E., Performance bounds for Level-Oriented Two-Dimensional Packing Algorithms, *SIAM J. Comput.* 9 (1980), 808-826.

[12] Garey M.R., Johnson D.S., "Strong" Np-Completeness Results, Motivation, Examples and Implications, *J. ACM* 25, no. 4, 1978, 499-508.

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

[14] Gilmore P.C., Gomory R.E., The theory and Computation of Knapsack Functions, *Operation Research* 15 (1967), 1045-1075.

[15] Gurel O., *Circular Grapf of Marker Layout*, IBM Data Processing Division, New York Scientific Center Report, no. 320-2965, Feb. 1969.

[16] Israni S.S., Sanders J.L., Two-Dimensional Cutting Stock Problem Research: Areview and a New Rećtangular Laypout Algorithm, *Journal of Manufacturing Systems* 1 (1982), 169.

[17] Israni S.S., Sanders J.L., Performance Testing of Rectangular Parts Nesting Heuristics, *Int. J. Prod. Res.*, 23 (1985), 437-456.

[18] Johnson D.S., *Near-Optimal Bin-Packing Algorithms*, Ph. D. Thesis, Massachusetts Institute of Technology, Electrical Department, 1974.

[19] Karp. R.M., Lenstra J.K., McDiarmid C.J.H., Rinnooy Kan A.H.G., Probabilistic Analysis of Combinatorial Algorithms: An Annotated Bibliography, in M.O'h Eigcarthaigh, J.K. Lenstra and A.H.G. Rinnooy Kan (eds.), *Combinatorial Optimization: Annotated Bibliographies*, J. Wiley, Chichester, 1984.

[20] Lenstra H.W.Jr., Integer Programming With a Fixed Number of Variables, *Math. Oper. Res.* 8, 1973, 538-548.

[21] Oliviera J.F., Ferreira J.S., *Solving Two-Dimensional Cutting Problems and Comparing Different Approaches*, unpublished paper, Instituto de Engenharia de Sistemas e Com -putadores, Largo Mompilher 22, 4000 Porto Portugal.

[22] Rinnooy Kan A.H.G., Probabilistic analysis of algorithms, *Annals of Discrete Mathematics* 31 (1987), 1-60.

[23] Silver E.A., Vidal R.V., de Werra D., A tutorial on heuristic methods, *European Journal of Operational Research* 5, 1980 153-162.

[24] Slominski L., Probabilistic analysis of combinatorial algorithms: a bibliography with selected annotation, *Computing* 28 (1982), 257-267.

[25] Ullman J.B., *The Performance of a Memory Allocation Algorithm*, Tech. Rept., no. 100, Princeton Univ., Electrical Engineering Department, 1971.

[26] Wang P.Y., Two Algorithms for Constrained Two-Dimensional Cutting Stock Problems, *Operation Research* 31 (1983), 573-586.

Received December 12, 1989