Comparing OR and CLP approaches to 2D angle cutting and packing problems

Tomasz Szczygieł

Institute of Automatic Control, Silesian University of Technology, Akademicka 16, 44-100 Gliwice, Poland, e-mail: tszczygiel@inetia.pl

Abstract. The paper compares two approaches to 2D angle cutting and packing problems: an operation research approach and constraint logic programming approach. The operation research approach is represented by integer linear programming and dynamic programming. The constraint logic programming approach is implemented in CHIP v.5.3. The application of both approaches to a specific example are presented.

1 Introduction

With packing or cutting problem we come across every day. The cutting problem consists of cutting a set of figures out of a larger figure with minimum losses. The packing problem consists of packing figures of given sizes into a larger figure without overlapping. Some of those problems may be modeled as simple figure (square or rectangle) cutting or packing problem. There are however problems that must be modeled as complex figure (e.g. angles) cutting or packing problem. Real-live problems in most cases are three-dimensional and only special cases may be modeled as two-dimensional (2D) cutting or packing problems. If the figure positions are fixed, we may talk about cutting or packing problems with no rotation. If the figure positions are not fixed, we have to consider cutting and packing problems with rotation. Generally, cutting and packing problems may be classified as knapsack problems, bin packing problems or puzzle problems (see[5]). They are classical combinatorial problems usually solved by Operation Research (OR) method: Integer Linear Programming (ILP) or Dynamic Programming (DP). They are known to belong to the class of NP-complete problems, [4] and [12]. A real-live packing problem examples given by packing (for transportation purposes) of high-current enclosed conductors and bus bars used in the power industry. The packing of those elements into containers may be modeled as a three-dimensional angle packing problem; however the problem of packing them into long-load trailer may best be modeled as two-dimensional angle packing problem. The solution must determine the co-ordinates of each figure foothold, i.e. the co-ordinates of a chosen point of the angle inside the larger figure. Obviously, the larger figure must have a discrete co-ordinate system. The first purpose of this paper is to develop known some known OR method for determining the footholds.. However, those methods are not user friendly and require complicated mathematical calcu-

lations. The second purpose of this paper is to present solutions for the same cutting and packing problems using Constrain Logic Programming (CLP). The great strengths of this approach is the declarativity: the final program is simple and short problem description.

The cutting or packing problems with rotation are more general then cutting or packing without rotation; in this paper small figures may be rotated by n*90°, n being a natural number.

The problem considered relies upon the discrete co-ordinate system from Fig.1.

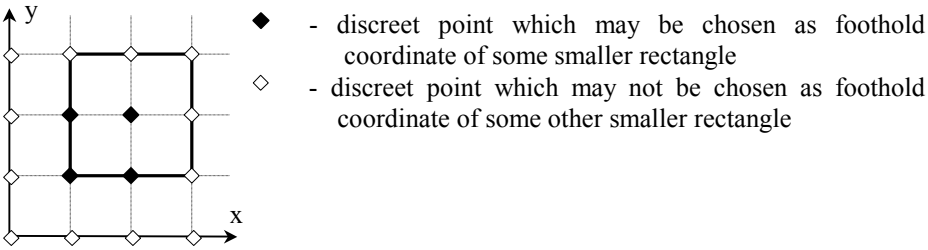


Fig.1. Discrete coordinate system for 2D figures

Each discrete location of the small figure in the large figure corresponds to a cost and is calculated as follows (see Fig.2) :

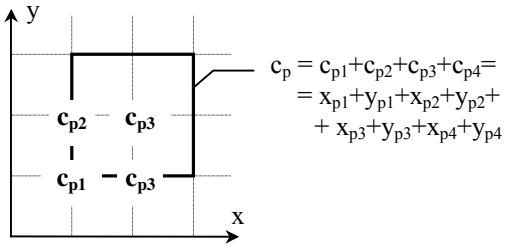


Fig.2. Cost associates with small figures dimensions.

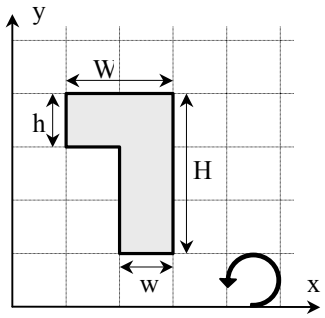


Fig. 3. Angle

Each angle may be described by a list including four characteristic sizes: (W, H, w, h) marked on the Fig.3. All possible lists of that kind may be found in [13].
To keep the presentation short, ILP, DP and CLP solutions for only one fairly general angle packing/cutting problem with rotation are presented

2 A ILP formulation for the Angle Cutting and Packing Problem

A global ILP problem formulation has been presented in [10] and [4]. The ILP two-dimensional angle packing problem formulation fits well into the standard ILP problem: Consider the problem of packing some small angles denoted by consecutive

number 1,2,...N_f. into a large rectangle. Let M={M1 | M2} where M1 is the set of all large rectangle co-ordinate pairs (there are M1 of them) and M2 is the set {1, 2,... m2.}. Let N={1, 2, …, n} where n is the total number of possible locations of all small angles in the large rectangle and let $P=\{P_j\}_{j\in N}$, be a set of subsets of M, such that each P_j has only one element from set M2 and any number of elements from M1.
Let $F_i=\{P_j: i\in P_j,\ j=1, 2, \ldots, n\}$ for each i=1, 2, …, m1+m2; therefore F_i consist of these sets P_j which include the *i-th* element of set M.
A zero-one variable x_j is defined as equal 1 if the *j*th member of F is selected and 0 if not. Let c_j denotes the cost associated with the *j*th member of F.
Let $A=[a_{ij}]$ denote the $m^x n$ incidence matrix of the members of F (columns A) versus the elements of M (rows A).

$$\text{Min } c^T x \tag{1}$$
$$Ax = e \tag{2}$$

for $\mathbf{x_j} = 0$ or 1 where j=1, 2, ..., *n*
and where **e** is vector of *m* ones.

$$m = (EndX * EndY) + N_f \tag{3}$$

$$n = \sum_{j=1}^{N_f} \left(K(EndX - W_j + 1)(EndY - H_j + 1)\right) + R \sum_{\forall j: W_j \neq H_j} \left(K(EndX - H_j + 1)(EndY - W_j + 1)\right) \tag{4}$$

where:

EndX – x axis size of larger figure
EndY – y axis size of larger figure
N_f – number of small figures
K = 1 for square, rectangle and angles without rotation
K = 4 for angles with rotation
Wj – width of *j–th* angle
H_j – height of *j–th* angle
R = 0 for no rotation figures
R = 1 for rotated figures

3 A DP formulation for the Angle Cutting and Packing Problem

Dynamic programming (DP) is a tool that can be used to solve many optimisation problems (see [2], [8], [11], [15]), because it transforms the problems into a series of smaller and more simple problems. This property makes possible the solution of diverse complex packing or cutting problems by transforming them into a multi-stage decision processes. This may be describing by the following mathematical formula for DP recursion:

$$f_n\begin{pmatrix} variations \\ h_1,...,\ h_n \end{pmatrix} = \min_{c_{h_n}} \left(c_{h_n} + f_{n-1}\begin{pmatrix} variations \\ h_1,...,\ h_{n-1} \end{pmatrix} \right) \tag{5}$$

where:

n – number of small figures

$h_1,\ldots,h_n$ – names of small figures

c_{hn} – cost of added figure h_n

f_n – minimum cost for n figures placed in a sequence $h_1,\ldots,h_n$.

This formula has to be supplemented by a constraint propagation formula, which decreases the number of coordinate points available as a result of the recent placement:

$$G_n\begin{pmatrix} variations \\ h_1,...,\ h_n \end{pmatrix} = G_{n-1}\begin{pmatrix} variations \\ h_1,...,\ h_{n-1} \end{pmatrix} - G_{\min\ c_{hn}}$$

$$G_n^T = \begin{bmatrix} g_{(0,0)} \\ \vdots \\ g_{(0,EndY\ )} \\ \vdots \\ g_{(EndX\ ,EndY\ )} \end{bmatrix} \tag{6}$$

$$G_0 = e$$

where:

n_c – number of discreet points of the large figure coordinate system

$g_{(x,y)} = 0$ – if the corresponding point of the large figure coordinate system is already covered by a small figure

$g_{(x,y)} = 1$ – if the corresponding point of the large figure coordinate system is not covered by a small figure

e – vector of n_c ones

EndX, EndY – sizes of the large figure

Number of all possible packing configuration without additional constraints is determinate by the permutations of all packing angles.

$$N = 16\ n_f\ ! \tag{7}$$

4 A CLP formulation for the Angle Cutting and Packing Problem

The problem discussed previously may by formulated also as constraint logic programming problem. The CLP formulation for the angle cutting/packing problem will be implemented in CHIP.
The idea of solving the angle packing problem with rotation using the classical *cumulative* global constraint is based on the fact that each angle is included in a rectangle. At the beginning all angles are divided into two rectangles (more details - see [13], [17]). This is done by the predicate *gen_rect/3* , defined as follows:

```
gen_rect([],[],[]).
gen_rect([D0L|D0T],[DH1,DH2|DT],[DX,DY|DXY]):-
        div_angle(D0L,DH1,DH2,DX,DY),
        gen_rect(D0T,DT,DXY).
div_angle([A,B,C,D],DH1,DH2,DX,DY):-
        DX  is C,
        DY  is B,
        E   is B-D,
        DH1 = C*D,
        DH2 = A*E.
```

Then those rectangles are placed with rotation onto the large figure; and additional constraint merges the two component rectangles into an angle. This is done by the predicates *constrain_rect/5* and *gen_lists/6* as follows:

```
constrain_rect([],[],[],[],[]).
constrain_rect([LXH1,LXH2|LXT],[LYH1,LYH2|LYT],
               [DXH1,DXH2|DXT],[DYH1,DYH2|DYT],[DX,DY|DT]):-
        dll_y(LYH1,LYH2,DYH1,DYH2,STY,ENY),
        dll_x(LXH1,LXH2,DXH1,DXH2,STX,ENX),
        [A,B] :: [DX,DY],
        (DX \= DY -> A #\= B; true),
        ENX #= A+STX,
        ENY #= B+STY,
        constrain_rect(LXT,LYT,DXT,DYT,DT).

dll_x(LXH1,LXH2,DXH1,DXH2,STX,ENX):-
        LXH1 #<= LXH2,
        LXH1+DXH1 #>= LXH2+DXH2,
        STX #= LXH1,
        ENX #= LXH1+DXH1.
dll_y(LYH1,LYH2,DYH1,DYH2,STY,ENY):-
        LYH1 #<= LYH2,
        LYH1+DYH1 #>= LYH2+DYH2,
        STY #= LYH1,
        ENY #= LYH1+DYH1.
```

Only one definition of predicates *dll_x/6* and *dll_y/6* is presented; other cases for this predicates may be defined in a similar way.

The *gen_lists/6* predicate is generated list of variable use in standard constraints - like *diffn* and *cumulative* - from data for problem. This predicate is as follows:

```
gen_lists([],[],[],[],[],[]).
gen_lists([X1*Y1,X2*Y2|T],[LXH1,LXH2|LXT],[LYH1,LYH2|LYT],
          [DXH1,DXH2|DXT],[DYH1,DYH2|DYT],[DFH1,DFH2|DFT]):-
        [DXH1,DYH1] :: [X1,Y1],
        [DXH2,DYH2] :: [X2,Y2],
        (X1 \= Y1 -> DXH1 #\= DYH1; true),
        (X2 \= Y2 -> DXH2 #\= DYH2; true),
        (DXH1 = X1 -> DXH2 #= X2; DXH2 #= Y2),
        (DYH1 = Y1 -> DYH2 #= Y2; DYH2 #= X2),
         append([LXH1,LYH1],[DXH1,DYH1],DFH1),
         append([LXH2,LYH2],[DXH2,DYH2],DFH2),
         gen_lists(T,LXT,LYT,DXT,DYT,DFT).
```

The final program solving this problem is as follows:

```
run:-
       data(Data0),
       gen_rect(Data0,Data,DXY),
       min_max((gen_lists(Data,LX,LY,DX,DY,DF),
                constrain_rect(LX,LY,DX,DY,DXY),
                diffn(DF,unused,unused,[EndX,EndY]),
         cumulative(LX,DX,DY,unused,unused,HighX,EndX,unused),
         cumulative(LY,DY,DX,unused,unused,HighY,EndY,unused),
                append(LX,LY,LXY),
                append(DX,DY,DDXY),
                append(LXY,DDXY,XY),
                labeling(XY)),EndX+EndY).
```

5 Computer experiment

Example I**:**
3 small angles are to be packed with rotation into a large rectangle so that none of them is overlapping any other. The sizes of larger rectangle are suitably: EndX=4 and EndY=4. Table 1 gives the data for the problem.

Table 1. Data for angle packing problem

n	$\mathbf{h_n}$	**List of angle sizes**
1	**1**	*[1,1,3,3]*
2	**2**	*[1,3,2,2]*
3	**3**	*[3,1,1,4]*

EndX = 4, EndY = 4, $N_f = 3$

Use (3) and (4) to determine the sizes of matrix A:
$m = (EndX * EndY) + N_f = (4*4) + 3 = 19$

$$n = \sum_{j=1}^{N_f} \left(K \left(EndX - W_j + 1 \right) \left(EndY - H_j + 1 \right) \right) =$$

$$= 4(4-3+1)(4-3+1) + 4(4-2+1)(4-3+1) + 4(4-4+1)(4-3+1) = 48$$

The set M is given by:

M1=[(0,0) (0,1) (0,2) (0,3) (1,0) (1,1) (1,2) (1,3) (2,0) (2,1) (2,2) (2,3) (3,0) (3,1) (3,2) (3,3)]

M2=[**h1**, **h2**, **h3**]

M=[**M1**|**M2**]

$$\mathbf{A1} = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\
1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\
1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
\quad
\mathbf{A3} = \begin{bmatrix}
1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\
1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\
1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1
\end{bmatrix}$$

$$\mathbf{A2} = \begin{bmatrix}
0&0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0\\
1&0&0&0&0&0&1&1&0&0&0&0&1&1&0&0&0&0&0&1&0&0&0&0\\
1&1&0&0&0&0&0&1&1&0&0&0&1&1&0&0&0&0&0&0&1&0&0&0\\
0&1&0&0&0&0&0&0&1&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\
1&0&0&0&0&0&1&0&0&1&0&0&1&0&1&0&0&0&1&0&0&1&0&0\\
1&1&1&0&0&0&1&1&0&1&1&0&1&1&1&1&0&0&1&1&0&0&1&0\\
1&1&1&1&0&0&0&1&1&0&1&1&0&1&1&1&0&0&0&1&1&0&0&1\\
0&1&0&1&0&0&0&0&1&0&0&1&0&0&0&1&0&0&0&0&1&0&0&0\\
0&0&1&0&0&0&1&0&0&1&0&0&0&0&1&0&1&0&1&0&0&1&0&0\\
0&0&1&1&1&0&0&1&0&1&1&0&0&0&1&1&1&1&1&1&0&1&1&0\\
0&0&1&1&1&1&0&0&1&0&1&1&0&0&0&1&1&1&0&1&1&0&1&1\\
0&0&0&1&0&1&0&0&0&0&0&1&0&0&0&0&0&1&0&0&1&0&0&1\\
0&0&0&0&1&0&0&0&0&1&0&0&0&0&0&0&1&0&0&0&0&1&0&0\\
0&0&0&0&1&1&0&0&0&0&1&0&0&0&0&0&1&1&0&0&0&1&1&0\\
0&0&0&0&1&1&0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&1&1\\
0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0
\end{bmatrix}$$

$$\mathbf{A} = [\mathbf{A1} \mid \mathbf{A2} \mid \mathbf{A3}]$$

$$\mathbf{x1} = \begin{bmatrix} x_1^1 & x_2^1 & \ldots & x_{16}^1 \end{bmatrix} \quad \mathbf{x2} = \begin{bmatrix} x_1^2 & x_2^2 & \ldots & x_{24}^2 \end{bmatrix} \quad \mathbf{x3} = \begin{bmatrix} x_1^3 & x_2^3 & \ldots & x_8^3 \end{bmatrix}$$

$$\mathbf{x} = [\mathbf{x1} \mid \mathbf{x2} \mid \mathbf{x3}]$$

$$\mathbf{c1} = [10\ 15\ 15\ 19\ 14\ 19\ 19\ 24\ 10\ 15\ 15\ 20\ 6\ 11\ 11\ 16]$$

$$\mathbf{c2} = [9\ 14\ 17\ 19\ 19\ 24\ 6\ 11\ 14\ 13\ 16\ 21\ 6\ 11\ 11\ 16\ 16\ 21\ 10\ 13\ 18\ 17\ 18\ 23]$$

$$\mathbf{c3} = [9\ 16\ 10\ 21\ 14\ 27\ 15\ 21]$$

$$\mathbf{c} = [\mathbf{c1} \mid \mathbf{c2} \mid \mathbf{c3}]$$

Thus, the optimal solution to the ILP problem is:

$$x\mathbf{1} = [0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0]$$

$$x\mathbf{2} = [0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]$$

$$x\mathbf{3} = [0\ 0\ 0\ 0\ 0\ 1\ 0\ 0]$$

Cost = 48 has the foothold co-ordinate: (0,0) for angle **h1**, (0,1) for angle **h2** and (3,0) for angle **h3** .

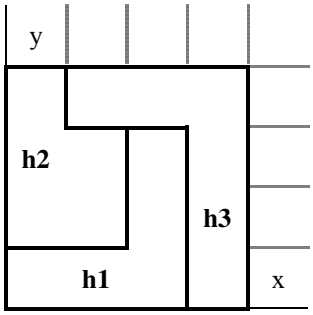
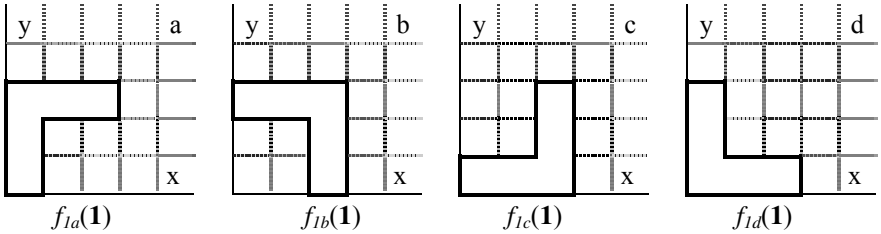


Fig. 3. A solution example I in ILP approach.

A DP approach to example I is given below.

Optimum solutions for a single placed angle are as follows:



$f_{1a}(\mathbf{1})$ $f_{1b}(\mathbf{1})$ $f_{1c}(\mathbf{1})$ $f_{1d}(\mathbf{1})$

Use (5) and (6) to calculate angle function:

a)

$f_{1a}(\mathbf{1}) = \min(c_{\mathbf{1a}} + f_0) = 10+0 = 10$

$G_{1a}(\mathbf{1}) = G_0 - G_{min\ c1a} =$

$= [1111111111111111] -$

$-[1110001000100000] =$

$= [0001110111011111]$

b)

$f_{1b}(\mathbf{1}) = \min(c_{\mathbf{1b}} + f_0) = 14+0 = 14$

$G_{1b}(\mathbf{1}) = G_0 - G_{min\ c1b} =$

$= [1111111111111111] -$

$-[0010001011100000] =$

$= [1101110100011111]$

c)

$f_{1c}(\mathbf{1}) = \min(c_{\mathbf{1c}} + f_0) = 10+0 = 10$

$G_{1c}(\mathbf{1}) = G_0 - G_{min\ c1c} =$

$= [1111111111111111] -$

$-[1000100011100000] =$

$= [0111011100011111]$

d)

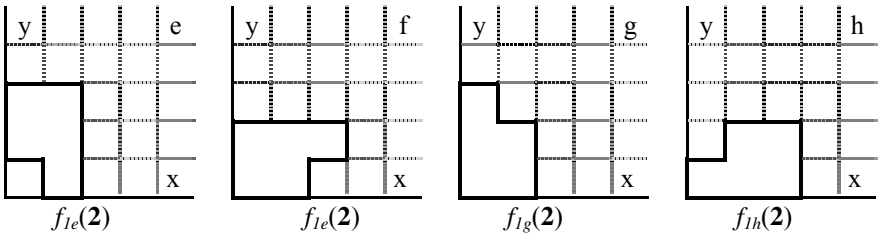
$f_{1d}(\mathbf{1}) = \min(c_{\mathbf{1d}} + f_0) = 6+0 = 6$

$G_{1d}(\mathbf{1}) = G_0 - G_{min\ c1d} =$

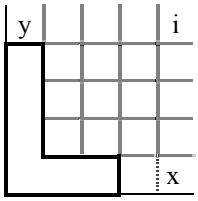
$= [1111111111111111] -$

$-[1110100010000000] =$

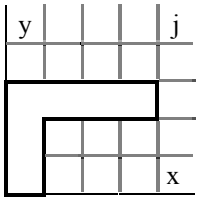
$= [0001011101111111]$ etc.



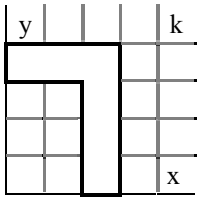
$f_{1e}(\mathbf{2})$ $f_{1e}(\mathbf{2})$ $f_{1g}(\mathbf{2})$ $f_{1h}(\mathbf{2})$



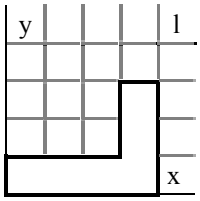
f_{1i}(**3**)



f_{1j}(**3**)

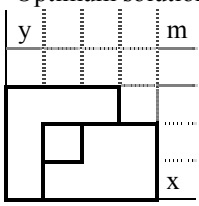


f_{1k}(**3**)

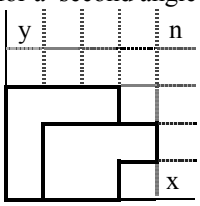


f_{1l}(**3**)

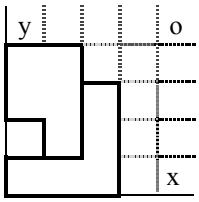
Optimum solutions for a second angle added are as follows:



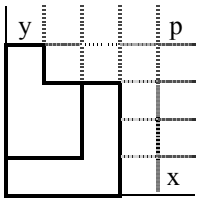
f_{2m}(**1,2**)



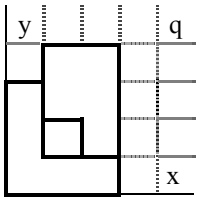
f_{2n}(**1,2**)



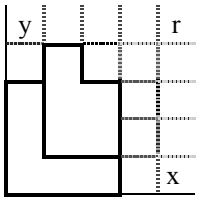
f_{2o}(**1,2**)



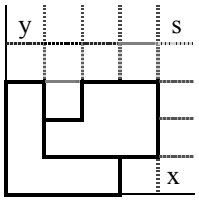
f_{2p}(**1,2**)



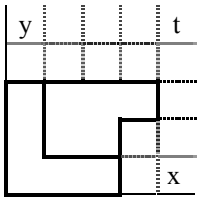
f_{2q}(**1,2**)



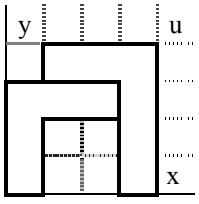
f_{2r}(**1,2**)



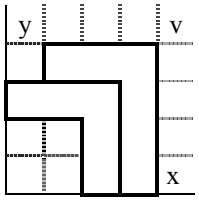
f_{2s}(**1,2**)



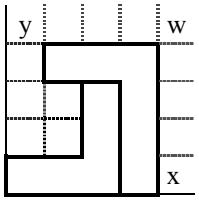
f_{2t}(**1,2**)



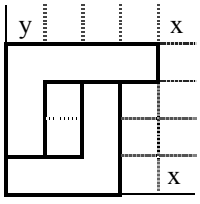
f_{2u}(**1,3**)



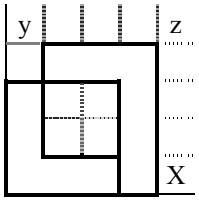
f_{2v}(**1,3**)



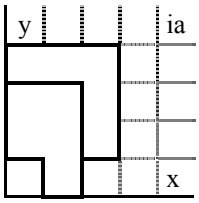
f_{2w}(**1,3**)



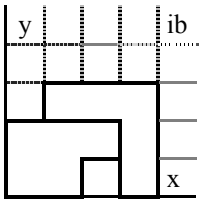
f_{2x}(**1,3**)



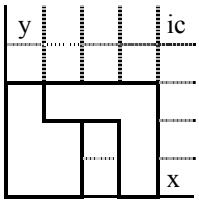
f_{2z}(**1,3**)



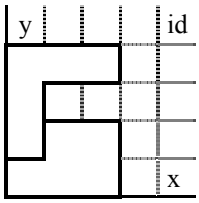
f_{2ia}(**2,1**)



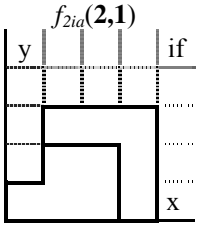
f_{2ib}(**2,1**)



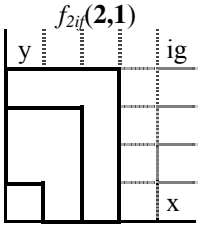
f_{2ic}(**2,1**)



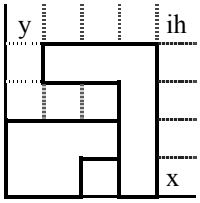
f_{2id}(**2,1**)



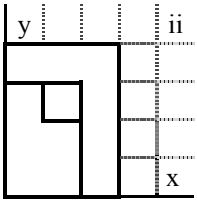
f_{2if}(**2,1**)



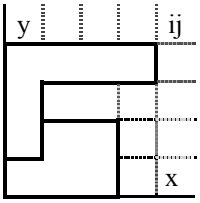
f_{2ig}(**2,3**)



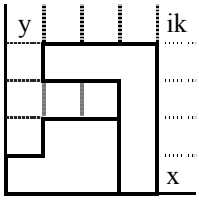
f_{2ih}(**2,3**)



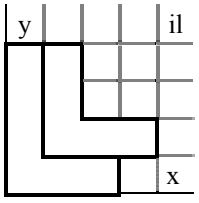
f_{2ii}(**2,3**)



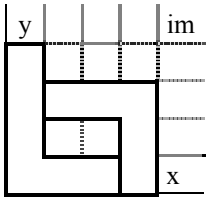
f_{2ij}(**2,3**)



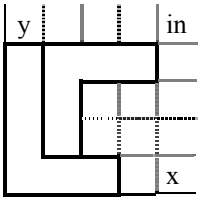
f_{2ik}(**2,3**)



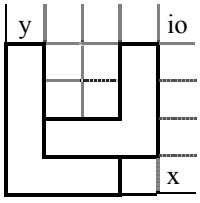
f_{2il}(**3,1**)



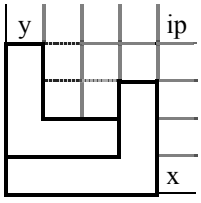
f_{2im}(**3,1**)



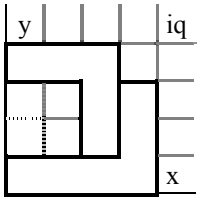
f_{2in}(**3,1**)



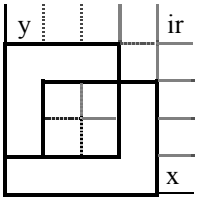
f_{2io}(**3,1**)



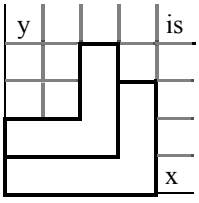
f_{2ip}(**3,1**)



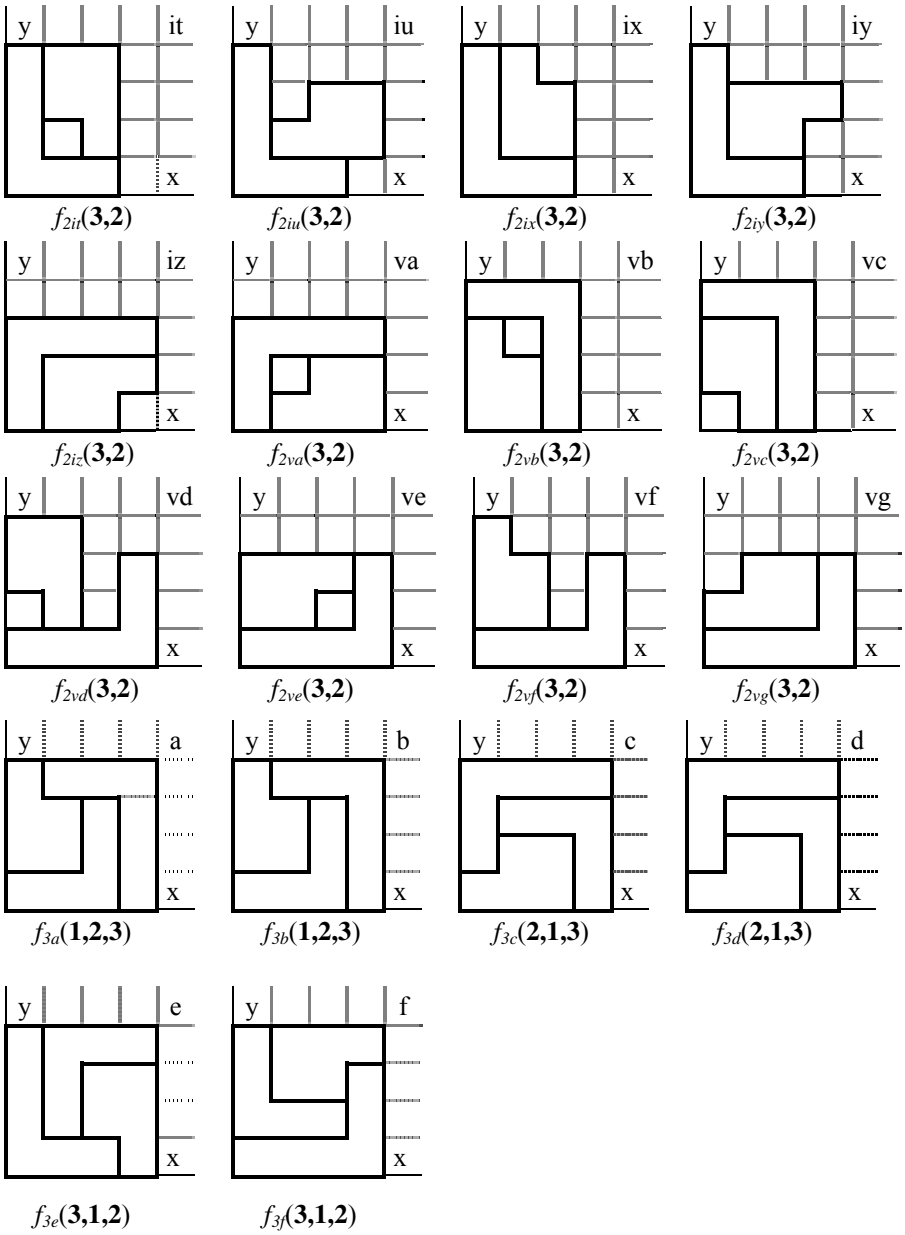
f_{2iq}(**2,3**)



f_{2ir}(**2,3**)



f_{2is}(**2,3**)



In this case, the optimal solution to the DP problem given by **Cost = 48** has the foot-hold co-ordinate: (0,0) for angle **h1**, (0,1) for angle **h2** and (3,0) for angle **h3** .

The CLP approach to Example I was presented in section 4. The CHIP model finds a solution (see Fig.4) after 0.731[s] on a Pentium II/300MHz, 64 MB station.

Fig. 4. A solution for the example I in CLP approach

6 A truly difficult example

To demonstrate additionally the effectiveness of CLP, it is worth presenting another example of the puzzle problem – the prefect square packing problem, see [1]. The solution presented bellow is however more simple and effective then the one from [1].

Example II**:**
Let EndX=EndY=112 be the size of the large square in which to pack 21 smaller squares, none of them overlapping any other. Table 2 gives the data for the problem.

Table 2. Data for perfect square packing problem.

j	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
$\mathbf{W_j}$	*2*	*4*	*6*	*7*	*8*	*9*	*11*	*15*	*16*	*17*	*18*	*19*	*24*	*25*
j	**15**	**16**	**17**	**18**	**19**	**20**	**21**							
$\mathbf{W_j}$	*27*	*29*	*33*	*35*	*37*	*42*	*50*							

If example II may be solved by Linear Programming sizes of matrix A_{m*n} are follows:

EndX = 112, EndY = 112, $N_f = 21$

$$m = (112 * 112) + 21 = 12565$$

$$n = \sum_{j=1}^{21} ((112 - W_j + 1)(112 - W_j + 1)) = 182609$$

The sizes of this problem make it impossible to use LINGO (LP solver) for solving it.

A dynamic programming approach to example II would call for solving of all possible packing configuration, their number being equal to:

$$N = n_f ! = 21! = 51090942171709440000 \approx 51*10^{18}$$

This clearly makes it impossible to use DP.

With help coming CLP. Using the *cumulative* and *diffn* CHIP standard global constraints, the heart of the model solving the perfect square packing problem is as follows:

```
solve_data(Data,EndXY,EndXY):-
        gen_lists(LX,LY,Data,End,Surface,DF),
        cumulative(LX,Data,Data,End,Surface,High,unused,unused),
        labeling(LX),
        diffn(DF,unused,unused,[EndXY,EndXY],unused,unused),
        labeling(LY),
        write(' LX: '),writeln(LX),
        write(' LY: '),writeln(LY).
```

Presented program is different from CHIP program discussed in point 4, the difference lies on the fact that now the packed figures are squares and this problem belongs to puzzle problems – problem without optimisation.

For perfect square packing problem, the CHIP model finds a solution (see Fig.5) after 1.9[s] on a Pentium II/300MHz, 64 MB station.

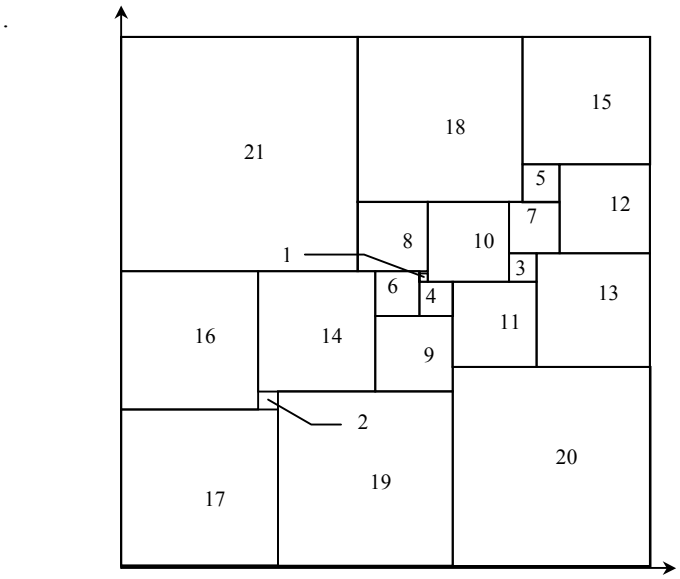


Fig. 5. A solution for the perfect square packing problem

8 Conclusions

For other cases: for rectangle and angle packing or cutting problem with no rotation, and three-dimensional angle packing or cutting problems the CLP formulation may bee found in [13], [14], [15], [16], [17].
The LP approach for packing or cutting problem requires a considerable amount of mathematical expertise whereas he CLP approach requires only a suitable declarative description.
The DP approach for this problem requires generally a big computational outlay and is not useful.
The presented results and detailed studies of angle packing or cutting problem demonstrate the effectiveness of CLP as a tool for fast solving complicated packing and cutting problems.

8 Acknowledgment

The author is grateful to Prof. A. Niederliński for his support, help and encouragement.
This research.was supported by a doctoral grant from the Polish Committee of Scientific Research (KBN) (BK-215/Rau-1/2001) and by a doctoral grant from the Foundation of Polish Science.

References

1. Aggoun A., Beldiceanu N.: Extending CHIP in order to solve complex scheduling and placement problems, Journal Mathematical and Computer Modeling Vol. 17, No. 7, 1993, 57-73
2. Bellman R., Dreyfus S.: Applied Dynamic Programming, Princeton University Press, Princeton, New Jersey, 1962
3. Bryant V.: Aspekty kombinatoryki (Aspects of Combinatorics), Wydawnictwa Naukowo-Techniczne Warszawa 1997
4. Christofides N., Mingozzi A., Toth P., Sandi C.: Combinatorial Optimization, John Wiley & Sons 1979
5. Dell'Amico M., Maffioli F., Martello S., Annotated bibliographies in combinatorial optimization, John Wiley & Sons, Chichester, New York, Weinheim, Brisbane, Toronto, Singapore 1997
6. Dincbas M., Simonis H., Van Hentenryck P.: Solving a Cutting-Stock Problem in Constraint Logic Programming, Logic Programming 1, Proc. 5 Inf. Conf. and Symp., MIT Press Cambridge, Massachusetts, London England 1998
7. Elion S., Christofides N.: The loading problem, Management Science Vol. 17, No. 4, 1971, 259-268
8. Gilmore P.C., Gomory R.E.: The theory and computation of knapsack functions, Operation Research Vol. 14, 1966, 1045-1079

9. Niederliński A.: Constraint logic programming - from Prolog to Chip, Proceedings of the CPDC'99 Workshop on Constraint Programming for Decision and Control, Gliwice, 1999, 27-34
10. Martello S., Toth P.: Knapsack problems, John Wiley & Sons, Chichester, New York, Brisbane, Toronto, Singapore 1990
11. Padberg M.W.: Covering, packing and knapsack problems, Annals of Discrete Mathematics Vol. 4, 1979, 265-287
12. Schrijver A,: Theory of linear and integer programming, John Wiley & Sons, Chichester, New York, Weinheim, Brisbane, Toronto, Singapore 1998
13. Szczygieł T.: Solving a two-dimensional angle packing problem in CHIP, Proceedings of the CPDC'99 Workshop on Constraint Programming for Decision and Control, Gliwice, 1999, 65-72
14. Szczygieł T.: Solving a three-dimensional angle packing problem in CHIP, Proceedings of the CPDC'2000 Workshop on Constraint Programming for Decision and Control, Gliwice, 2000, 59-66
15. Szczygieł T.: Solving angle packing problems. Dynamic Programming vs. Constraint Logic Programming, Proceedings of the CPDC'2001 Workshop on Constraint Programming for Decision and Control, Gliwice, 2001, 53-60
16. Szczygieł T.: CLP approaches to 2D angle placements, Proceedings of the 2001 ERCM Workshop on Constraint, Prague, 2001,
17. Szczygieł T.: Rozwiązywanie problemów upakowania na płaszczyźnie i w przestrzeni, (Solving a two-dimensional and three-dimensional packing problem) Zeszyty Naukowe Politechniki Śląskiej 2000 – seria Automatyka z.131, Gliwice, 2000