# CP Model - Present Wrapping Problem

Filippo Lo Bue

September 2020

## 1 Description of the Problem

Given a wrapping paper roll of a certain dimension and a list of presents, decide how to cut off pieces of paper so that all the presents can be wrapped. Consider that each present is described by the dimensions of the piece of paper needed to wrap it. Moreover, each necessary piece of paper cannot be rotated when cutting off, to respect the direction of the patterns in the paper.

The purpose of this project is to solve PWP proceeding as follows:

1. Start with the variables and the main problem constraints.
2. In any solution, if we draw a vertical line and sum the vertical sides of the traversed pieces, the sum can be at most $l$.
3. Use global constraints to impose the main problem constraints and the implied constraints in your CP model.
4. Investigate the best way to search for solutions in CP.
5. If rotation was enabled? which of the CP model encoding is easier to modify to take this into account? How would you modify that model/encoding?
6. there can be multiple pieces of the same dimension: how would you improve the CP model encoding?

Points **1 - 4** are part of the MiniZinc program.
Point **5** is expressed using mathematical notation and it was included in a second MiniZinc program.
Point **6** is expressed using mathematical notation and it was included in a third MiniZinc program.

# 2 pwp_v9.mzn (Points 1-4)

## 2.1 Parameters

- **pr_w**: width of the wrapping paper roll.
- **pr_h**: height of the wrapping paper roll.
- **n_pieces**: number of the presents/pieces.
- **L**: dimensions of the piece of paper needed to wrap each presents.
- **index_largest_p**: index of the biggest present/piece by area.
- **widths_set_values**: distinct(set) pieces' widths
- **ordered_widths_values**: widths_set_values descending ordered
- **group_number**: number of different widths(cardinality of widths_set_values).
- **pieces_group_by_w**: array of set pieces' indexes grouped by widths.
- **number_columns_per_group**: number of columns needed for each group of pieces.
- **independent_solving_on_w**: If 'true' means we renounce to the generality of the model (so it is not important to know the exact number of total solutions anymore), we want to speed up the process of finding 1 solution.

## 2.2 Variables

$$\begin{aligned}\mathbf{q} &= [[piece\_1\_x, piece\_1\_y] \\ &= [piece\_2\_x, piece\_2\_y] \\ &\ \ ... \\ &= [piece\_n\_x, piece\_n\_y]]\end{aligned}$$

Bottom left corner of each pieces

## 2.3 Constraints

### 2.3.1 the Double Cumulative Constraint(Point 2)

The cumulative constraint is not only used for scheduling tasks but it is used also for spatial positioning of objects(the presents) inside a container(paper roll). Essentially We can see every presents/pieces as an action which need to be scheduled, the x coordinate as the starting time of that action, the width as the duration and the height as the resources consuming.

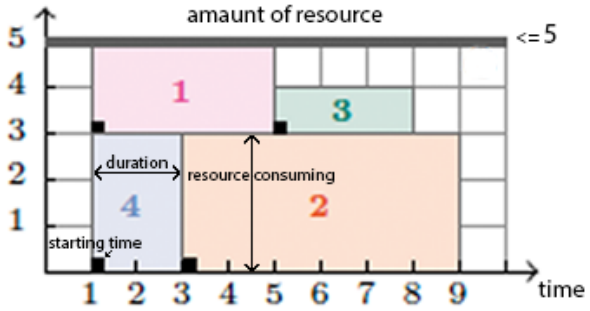



Figure 1: "Horizontal" cumulative constraint

The second cumulative constraint has the same application as the first inverting the axes and the meaning of the actions'(pieces') duration(now the heights) and the resource consuming(now the widths).

```
% --- Point 2 ---
constraint cumulative(q[PIECES,1], L[PIECES, 1], L[PIECES, 2], pr_h);
constraint cumulative(q[PIECES,2], L[PIECES, 2], L[PIECES, 1], pr_w);
```

### 2.3.2 No exceeding the paper dimensions

No piece must have any part outside the paper roll.

```
constraint forall (i in PIECES) (
          (q[i,1] + L[i, 1]) <= pr_w /\ (q[i,2] + L[i, 2]) <= pr_h
      );
```

### 2.3.3 No-overlap

No object must ever overlap with any other object.

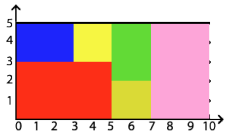
```
constraint diffn_k(q, L);
```

### 2.3.4 Symmetry breaking rules[2]

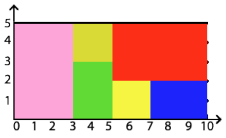
The Problem so far contains a number of symmetries, which we need to remove as we may have to explore the complete search space. The symmetries that we want to remove are the horizontal reflection, vertical reflection and a combination of both. Considering a generic solution:


Figure 2: generic solution

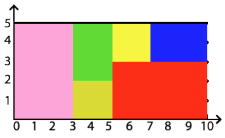
We have the following 3 "reflected" solutions:



(a) horizontal symmetry



(b) vertical symmetry



(c) double symmetry

Figure 3: Reflected solutions

To remove those we can restrict the domain of one present in an enclosing rectangle of size $Width \times Height$ to:

$$X :: 1..1 + \lfloor \frac{Width - w\_present}{2} \rfloor,\ Y :: 1..1 + \lfloor \frac{Height - h\_present}{2} \rfloor$$

Considering we can choose any present, it is better to restrict the domain of the **largest** one of size $w\_max \times h\_max$ for ensuring a much bigger domain reduction propagation for the others presents.

$$X :: 1..1 + \lfloor \frac{Width - w\_max}{2} \rfloor,\ Y :: 1..1 + \lfloor \frac{Height - h\_max}{2} \rfloor$$

```
int: index_largest_p = arg_max([ L[i,1]*L[i,2] | i in PIECES]); % index
    of the largest piece
...
constraint
       2 * (q[index_largest_p,1]) <= (pr_w - L[index_largest_p, 1])
   /\  2 * (q[index_largest_p,2]) <= (pr_h - L[index_largest_p, 2]);
```

### 2.3.5 Independent Solving based on widths

Constraint applied only if *independent_solving_on_w* is *true*(only on the $x$ axis). $\forall$ group in *pieces_group_by_w* the left margin(the starting point for that group) is calculated and all the pieces in that group will be constrained to be between the $left\_margin$ and $left\_margin + (columns - 1) * width$ (maximum width that the columns occupy for that group). this constraint allows us to speed up the resolution of many otherwise unsolvable instances.

E.g. Instance $18 \times 18$

**pr_w**=18, **pr_h**=18, **n_pieces**=16

$$\begin{aligned}\mathbf{L} &= [[3,3],[3,4],[3,5],\ [3,6]\\ &= \ [3,7],[3,8],[3,10],[3,11]\\ &= \ [4,3],[4,4],[4,5],\ [4,6]\\ &= \ [5,3],[5,4],[5,5],\ [5,6]]\end{aligned}$$

**pieces_group_by_w**=[13..16, 9..12, 1..8] pieces with the same width(5) from the $13^{th}$ to the $16^{th}$ array's element, pieces with width equals to 4 from the $9^{th}$ to the $12^{th}$ array's element and pieces with width equals to 3 from the $1^{st}$ to the $8^{th}$ array's element.

**number_columns_per_group**=[1, 1, 3] only one column is required for pieces with width 5 and 4(perfect stacking), instead for the pieces with width 3, three columns are needed.

So:
**independent_solving_on_w** = *true* because $\forall$*pieces_group_by_w* the sum of the pieces' heights are greater or equal then 18.

$$\begin{aligned}&width\ 5 && : 3+4+5+6 && = 18\\ &width\ 4 && : 3+4+5+6 && = 18\\ &width\ 3 && : 3+4+5+6+8+8+10+11 && = 54\end{aligned}$$

So:
*independent_solving_on_w = true*
**pieces_group_by_w**=[13..16, 9..12, 1..8]
**number_columns_per_group**=[1, 1, 3]

**left_margin**

$$\begin{aligned}&\text{for the } 1^{st} \text{ group(width 5)} && = 0\\ &\text{for the } 2^{nd} \text{ group(width 4)} && = number\_columns\_per\_group[1] \times total\_group\_width[1] = 3\\ &\text{for the } 3^{rd} \text{ group(width 3)} && = number\_columns\_per\_group[1] \times total\_group\_width[1] +\\ & && \quad number\_columns\_per\_group[2] \times total\_group\_width[2] = 9\end{aligned}$$

$\forall$*pieces with index* $\in [13..16] : x\_coordinate \geq \mathbf{0}$(left_margin $1^{st}$ group) &
$x\_coordinate \leq \mathbf{0}$(left_margin $1^{st}$ group) + 0 * 5(columns-1)*width

$$\forall \textit{pieces with index} \in [9..12] : x\_coordinate \geq \textbf{3}(\text{left\_margin } 2^{st} \text{ group}) \;\&\; x\_coordinate \leq \textbf{3}(\text{left\_margin } 1^{st} \text{ group}) + 0 * 4(\text{columns-1})*\text{width}$$

$$\forall \textit{pieces with index} \in [1..8] : x\_coordinate \geq \textbf{9}(\text{left\_margin } 2^{st} \text{ group}) \;\&\; x\_coordinate \leq \textbf{15}(\text{left\_margin } 1^{st} \text{ group}) + 2 * 3(\text{columns-1})*\text{width}$$

```
% Independent Solving based on widths
predicate com_prec(int: i, int: left_margin) = let {
    int: n_pieces = card(pieces_group_by_w[i]);
    array[1..n_pieces] of PIECES: indexes = [j | j in
        pieces_group_by_w[i]];
    int: columns = number_columns_per_group[i];
    int: width = total_group_width[i];
  } in forall(h in 1..n_pieces)
    (q[indexes[h],1] >= left_margin /\
     q[indexes[h],1] <= left_margin+(columns-1)*width);

constraint (independent_solving_on_w) ->
       forall(i in 1..group_number) (com_prec(i, group_starting_w(i)));
```

# 3 pwp_v9-rot.mzn (Point 5)

The same previous model with handling the possible rotation of each presents/-pieces without the *independent solving based on widths* constraint.

## 3.1 New Variables

**rot**: Array of 0|1 with dimension equal to the number of presents/pieces. If the $i^{th}$ element of the array is *'0'* means no rotation otherwise if *'1'* that piece is rotated by 90°

$$\textbf{rot} \quad = [rot\_1, rot\_3, \; \ldots \, , rot\_n]$$

## 3.2 Relevant Function

$function$ **int** : $get\_dim(int : i, int : d)$ : According to the rotation, return the correct value of the width($d$=*1*)/height($d$=*2*) of a given piece index($i$). This function is used every time I need to know the width/height of a piece, the pieces's dimension is not constant anymore like in the previous model.

## 3.3 New Constraints

In all constraints we no longer use the initial dimensions of the pieces but we use the new function *get_dim*.

```
% --- Point 2 ---
constraint cumulative(q[PIECES,1], [ get_dim(i, 1) | i in PIECES ],
                        [ get_dim(i, 2) | i in PIECES ], pr_h);
constraint cumulative(q[PIECES,2], [ get_dim(i, 2) | i in PIECES ],
                        [ get_dim(i, 1) | i in PIECES ], pr_w);

%-No exceeding the paper dimensions
constraint forall (i in PIECES)
                  (   (q[i,1] + get_dim(i, 1)) <= pr_w /\
                      (q[i,2] + get_dim(i, 2)) <= pr_h);

% Non-overlap
function array[PIECES] of var int:
          get_q(int: d) = [q[i, d] | i in PIECES];
function array[PIECES] of var int:
          L_axis(int: d) = [get_dim(i,d) | i in PIECES];

constraint diffn(get_q(1),get_q(2),L_axis(1),L_axis(2));

%Symmetry breaking rules
constraint
   2 * (q[index_largest_p,1]) <= (pr_w - get_dim(index_largest_p, 1)) /\
   2 * (q[index_largest_p,2]) <= (pr_h - get_dim(index_largest_p, 2));
```

#### 3.3.1 square = no rotation

It is unnecessary to rotate any square piece (width = height) because I will get the same solution (same coordinates for each piece).

```
constraint forall (i in PIECES where L[i,1]=L[i,2]) (rot[i] = 0);
```

# 4 pwp_v9-same-dim.mzn (Point 6)

The same model as *pwp_v9.mzn* but taking into consideration the fact that there can be multiple pieces with the same dimension. Also the previous models are able to solve instances in which there are several equal pieces but in those cases it is possible to fix the positional relation with each other reducing the domains for some pieces and speeding up the instance resolution itself[3].

## 4.1 New Constraints

### 4.1.1 same dimension

If we have rectangles/pieces $r_i$, $r_j$ and $r_k$ which have the same dimensions we can fix/constrain the positional relation of those rectangles, between $r_i$-$r_j$ and

$r_j$-$r_k$. For example, if we have 2 pieces with the same dimension($2 \times 2$), the second piece(the pink one) will never placed to the left or under the first one(the yellow one). it will be placed always on the right or upper the first one.

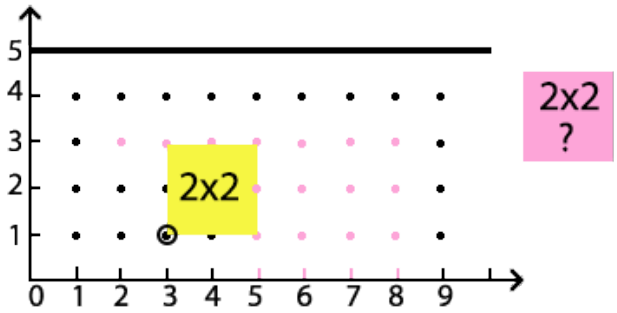



Figure 4: Positional relation constraint

```
constraint
    forall( i, j in PIECES
           where i < j /\ L[i,1]==L[j,1] /\ L[i,2]==L[j,2])
    ( lex_less(q[i,1..2] , q[j,1..2]) );
```

# 5 Experiments Results

In this section will be shared all the relevant experiments results contained in the jupyter notebook *PWP.ipynb*.

## 5.1 TEST 1

Number of solutions (#) and failures when looking for all solutions to some instances of the PW Problem, comparing the ordering of pieces or not, using the default search.

Independent solving on width deactivated('bool: independent_solving_on_w = **false**;'), No search_ann and No "**Symmetry breaking rules**" part used.

| n | # | $no-ord\_time$ | $no-ord\_failures$ | $ord\_time$ | $ord\_failures$ | $decr-ord\_time$ | $decr-ord\_failures$ |
|---|---|---|---|---|---|---|---|
| 8x8 | 12 | 0 | 2 | 0 | 2 | 0 | 3 |
| 9x9 | 24 | .001 | 4 | .001 | 4 | .001 | 21 |
| 10x10 | 64 | .002 | 58 | .003 | 58 | .003 | 58 |
| 11x11 | 128 | .011 | 422 | .011 | 426 | .014 | 554 |
| 12x12 | 192 | .013 | 764 | .014 | 753 | .016 | 1096 |
| 13x13 | 1568 | .115 | 3421 | .109 | 3368 | .115 | 3674 |
| 14x14 | 1344 | .082 | 1110 | .082 | 1149 | .099 | 2602 |
| 15x15 | 10752 | .766 | 9068 | .772 | 9068 | 1.086 | 23246 |
| 16x16 | 2304 | .162 | 2170 | .171 | 2149 | .218 | 2052 |
| 17x17 | 97856 | 14.64 | 237,574 | 12.897 | 177,718 | 35.886 | 980848 |
| 18x18 | / | - | - | - | - | - | - |
| 19x19 | / | - | - | - | - | - | - |
| 20x20 | 566784 | 77.84 | 2,681,267 | 85 | 2,626,486 | - | - |
| 21x21 | / | - | - | - | - | - | - |

Now the same test handling all the possible **symmetries**.

| n | # | $no-ord\_time$ | $no-ord\_failures$ | $ord\_time$ | $ord\_failures$ | $decr-ord\_time$ | $decr-ord\_failures$ |
|---|---|---|---|---|---|---|---|
| 8x8 | 0 | 0 | 2 | 0 | 2 | 0 | 3 |
| 9x9 | 0 | .001 | 4 | .001 | 4 | .001 | 21 |
| 10x10 | 22 | .001 | 58 | .003 | 58 | .003 | 58 |
| 11x11 | 32 | .003 | 422 | .011 | 426 | .014 | 554 |
| 12x12 | 48 | .004 | 764 | .014 | 753 | .016 | 1096 |
| 13x13 | 532 | .047 | 3421 | .109 | 3368 | .115 | 3674 |
| 14x14 | 336 | .022 | 1110 | .082 | 1149 | .099 | 2602 |
| 15x15 | 6272 | .438 | 9068 | .772 | 9068 | 1.086 | 23246 |
| 16x16 | 576 | .038 | 2170 | .171 | 2149 | .218 | 2052 |
| 17x17 | 24464 | 2.576 | 237,574 | 12.897 | 177,718 | 35.886 | 980848 |
| 18x18 | / | - | - | - | - | - | - |
| 19x19 | 1,383,360 | 185.543 | 5,231,506 | 182.956 | 5,233,576 | - | - |
| 20x20 | 141696 | 15.854 | 434,603 | 17.104 | 434,203 | 20.927 | 751.911 |
| 21x21 | 478656 | 76.467 | 1,985,506 | 79.032 | 1,972,509 | 79.75 | 2,219,187 |

With the default search the best model is: incremental order pieces

## 5.2 TEST 2

Number of failures when looking for all solutions(with symm breaking) using different search heuristics (managed the descending order inside the model based on the area value).

The best results are marked with a "*"
**bold** number: time limit reached(5 min.)

| | Input-min | | ff-min | | DomWdeg-min | |
|---|---|---|---|---|---|---|
| n | *ord* | *decr − ord* | *ord* | *decr − ord* | *ord* | *decr − ord* |
| 8x8 | 0* | 0* | 2 | 2 | 2 | 2 |
| 9x9 | 3* | 3* | 3* | 3* | 3* | 3* |
| 10x10 | 8* | 8* | 8* | 8* | 10 | 10 |
| 11x11 | 46 | 46 | 43* | 43* | 49 | 49 |
| 12x12 | 70 | 70 | 60* | 60* | 96 | 96 |
| 13x13 | 1064* | 1079 | 1075 | 1073 | 1120 | 1120 |
| 14x14 | 448 | 445 | 265 | 269 | 262 | 261* |
| 15x15 | 4029* | 4354 | 5098 | 5039 | 5073 | 4997 |
| 16x16 | 583 | 573 | 583 | 573 | 524* | 588 |
| 17x17 | 75,311 | 76,210 | 50,606 | 51,214 | 49,866* | 50,859 |
| 18x18 | **11,925,446** | **11,935,654** | **9,123,160** | **9,191,395** | **8,695,525** | **8,633.719*** |
| 19x19 | 3,663,131 | 3,666,035 | 3,209,023* | 3,211,247 | 3,214,267 | 3,213,256 |
| 20x20 | 430,396 | 370,630 | 447,872 | 371,263 | 434,203 | 371,235* |
| 21x21 | 1,719,412-70 | 1,721,371-71 | 1,419,547-69 | 1,437,661-66 | 1,418,944* − 68 | 1,436,707-69 |

failures-time[$s$]

| | Input-rand | | ff-rand | | DomWdeg-rand | |
|---|---|---|---|---|---|---|
| n | *ord* | *decr − ord* | *ord* | *decr − ord* | *ord* | *decr − ord* |
| 8x8 | 1 | 1 | 2 | 2 | 2 | 2 |
| 9x9 | 4 | 4 | 4 | 4 | 4 | 4 |
| 10x10 | 11 | 11 | 12 | 12 | 12 | 12 |
| 11x11 | 48 | 48 | 44 | 44 | 47 | 45 |
| 12x12 | 100 | 100 | 77 | 77 | 90 | 90 |
| 13x13 | 1168 | 1130 | 1171 | 1141 | 1174 | 1195 |
| 14x14 | 493 | 501 | 315 | 313 | 300 | 316 |
| 15x15 | 4929 | 4834 | 5481 | 5457 | 5471 | 5441 |
| 16x16 | 581 | 587 | 581 | 587 | 593 | 582 |
| 17x17 | 84,216 | 85,261 | 54,832 | 54,260 | 54,970 | 56,626 |
| 18x18 | **10,751,176** | **10,834,891** | **11,986,136** | **11,531,960** | **12,141,790** | **11,018,706** |
| 19x19 | 4,088,991 | 4,137,855 | 3,608,068 | 3,635,708 | 3,587,009 | 3,597,849 |
| 20x20 | 467,476 | 420,972 | 478,043 | 434,628 | 470,117 | 408,973 |
| 21x21 | 1,941,636-77 | 1,949,438-74 | 1,587,390-65 | 1,609,090-67 | 1,590,398-65 | 1,607,347-67 |

failures-time[$s$]

The best search strategy is the $DomWdeg\_min$.

NB: The search heuristics are always based on the descending order of all the pieces' area, so the input order is not so relevant such as for the default search(TEST 1).

## 5.3 TEST 3

Time and failures to find the first solution for each instance.
Independent solving on width activated.

| | Input-min | | ff-min | | DomWdeg-min | |
|---|---|---|---|---|---|---|
| n | Time[s] | failures | Time[s] | failures | Time[s] | failures |
| 8x8 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9x9 | 0 | 0 | .001 | 0 | .001 | 0 |
| 10x10 | 0 | 0 | 0 | 0 | 0 | 0 |
| ... | ... | ... | ... | ... | ... | ... |
| 22x22 | .001 | 0 | .001 | 6 | .001 | 6 |
| 23x23 | 7.543 | 298779 | 10.191 | 398237 | 1.014 | 36927 |
| 24x24 | .001 | 24 | 0 | 0 | .001 | 0 |
| ... | ... | ... | ... | ... | ... | ... |
| 37x37 | .352 | 7400 | .156 | 3594 | .307 | 6343 |
| 38x38 | .025 | 595 | .002 | 21 | .001 | 6 |
| 39x39 | .003 | 0 | .002 | 0 | .002 | 0 |
| 40x40 | .001 | 14 | .002 | 14 | .001 | 1 |
| Tot: | **8.244s** | **312,514** | 24.817s | 947,511 | 18.989s | 672,773 |

In **bold** the best result. The *Input-min* is the best strategy.
All the instances are solved in 8.2 seconds and the 23x23 instance is the most difficult one to solve.

NB: the total is calculated considering all the instances. Open the report.odt file for the full test results.

# 6 References

[1] Lesh, N., Marks. J., McMahon A., Mitzenmacher M. (2004) *Exhaustive approaches to 2D rectangular perfect packings.* https://www.eecs.harvard.edu/ michaelm/postscripts/ipl2004.pdf

[2] Simonis, H. & O'Sullivan, B. (2008) *Search Strategies for Rectangle Packing.* International Conference on Principles and Practice of Constraint Programming: pp 52-66.

[3] Takehide, S., Katsumi, I., Naoyuki, T., Mutsunori, B., Hidetomo, N. *A SAT-based Method for Solving the Two-dimensional Strip Packing Problem.* http://ceur-ws.org/Vol-451/paper16soh.pdf

[4] Clautiaux, F., Jouglet, A., Carlier, J., Moukrim, A. (2006) *A new constraint programming approach for the orthogonal packing problem.* http://vmk.ugatu.ac.ru/c%26p/article/new_2009/2D_OPP_clautiaux_constraint_progr.pdf

[5] Huang, E., Korf, R.E. (2012) *Optimal Rectangle Packing: An Absolute Placement Approach.* https://arxiv.org/ftp/arxiv/papers/1402/1402.0557.pdf