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_v8.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. (See Functions)

2.2 Variables

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

Bottom left corner of each pieces

2.3 Functions

function **bool** : *independent_solving_on_w_possible*(): *true* if the sum of the pieces' height contained in each pieces_group_by_w is greater or equal then **pr_h**. This is sufficient for the following consideration: I can constrain each group of pieces to be placed in a specific, narrower piece of paper roll speeding up the instance resolution. (See the next section for more detail)

E.g. Instance 18×18

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.

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

2.4 Constraints

2.4.1 the double cumulative (lines 57,58)

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.

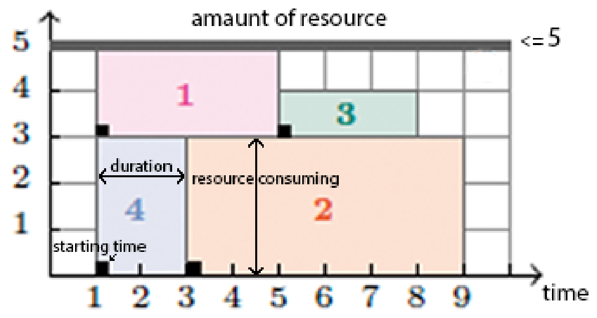

Figure 1: "Horizontal" cumulative constraint

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).

2.4.2 No exceeding the paper dimensions (line 37)

No piece must have any part outside the paper roll.

2.4.3 No-overlap (lines 40-46)

No object must ever overlap with any other object.

2.4.4 Symmetry breaking rules[2] (lines 52-54)

We restrict the domain of the largest square of size $w\_max \times h\_max$ to be placed in an enclosing rectangle of size $Width \times Height$ to:

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

2.4.5 Implied constraint (lines 57,58)

In any solution, if we draw a vertical line and sum the vertical sides of the traversed pieces, the sum can be at most n(number of presents/pieces). A similar property holds if we draw a horizontal line.

2.4.6 Independent Solving based on widths (lines 62-69)

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.

let's go back to the example above, the instance 18×18:

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

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

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

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

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

3 pwp_v8-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°

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

3.2 New 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 I no longer use the initial dimensions of the pieces but I use the new function *get_dim*.

3.3.1 square = no rotation (lines 40, 41)

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

4 pwp_v8-same-dim.mzn (Point 6)

The same model as *pwp_v8.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 pieces with the same dimensions 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 (lines 40-42)

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(2×2), 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.

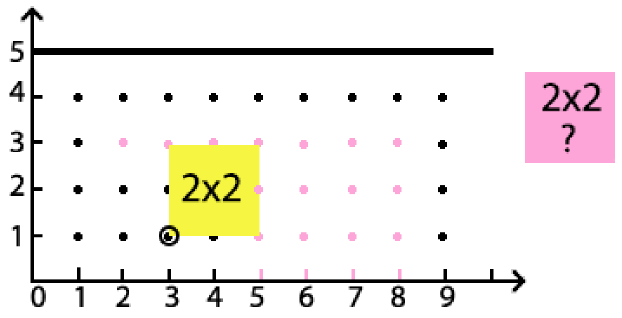

Figure 2: Positional relation constraint

5 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