Marco Buiani
Daniele Verì

Present Wrapping Problem CP MODEL

Abstract

The Present Wrapping problem can be viewed as finding the position of rectangular pieces in order to fit them into the available rectangular paper shape.
Moreover, the presents cannot be rotated nor fall outside of the bounding paper, not even partially.
In this report we present our MiniZinc Constraint Programming modelling work to face the problem.

Decision variables

The present wrapping problem provides the following input parameters:

- Paper width: W
- Paper heigth: H
- Number of presents: N
- Width and heigth of each k present, respectively $s_{k\,x}$ and $s_{k\,y}$.

We use an $N \times 2$ shaped array of integer decision variables to represent the 2D position of each present.
The bottom-left corner coordinates of a generic k present are represented by $p_{k\,x}$ and $p_{k\,y}$ variables

Available-paper bounding constraint

The available paper shape constrains the position of present rectangles to be inside the bounding box:

$$\forall k \in N,\ 0 \leqslant p_{k\,x} \leqslant W - s_{k\,x}$$
$$\forall k \in N,\ 0 \leqslant p_{k\,y} \leqslant H - s_{k\,y}$$

Non-overlapping constraint

No present can overlap with one another, this is imposed between each present pair:

$$\forall k_1, k_2 \in N,$$
$$p_{k_1\,x} + s_{k_1\,x} \leqslant p_{k_2\,x} \vee\ p_{k_1\,x} \geqslant p_{k_2\,x} + s_{k_2\,x} \vee$$
$$p_{k_1\,y} + s_{k_1\,y} \leqslant p_{k_2\,y}\ \vee\ p_{k_1\,y} \geqslant p_{k_2\,y} + s_{k_2\,y}$$

This is expressed by using the `diffn` MiziZinc global, specifically built to constrain rectangular shapes to not overlap.

```
constraint diffn(present_pos[1..n,1], present_pos[1..n,2],
                 present_shape[1..n,1], present_shape[1..n,2]);
```

Where:

- `present_pos[1..n,1]` is $(p_{1\,x}\,,\,..\,,\,p_{N\,x})$,
- `present_pos[1..n,2]` is $(p_{1\,y}\,,\,..\,,\,p_{N\,y})$,
- `present_shape[1..n,1]` is $(s_{1\,x}\,,\,..\,,\,s_{N\,x})$,
- `present_shape[1..n,2]` is $(s_{1\,y}\,,\,..\,,\,s_{N\,y})$

Partial sums implied contraint

At each moment during search, no column or row can be assigned to more than its total capacity.
We add the suggested partial sums implied constraint for each row and column.

If the present k occupies the row r, **then** $row_{k\,r}$ value is the heigth of the present, **else** 0.
If the present k occupies the column c, **then** $col_{k\,c}$ value is the width of the present, **else** 0.
Expressed in First Order Logic:

$$(p_{k\,x}\ \leqslant\ r\ \wedge\ p_{k\,x} + s_{k\,x} > r\ \implies row_{k\,r} =\ s_{k\,y})\ \wedge\ (p_{k\,x}\ >\ r\ \vee\ p_{k\,x} + s_{k\,x} \leqslant r\ \implies row_{k\,r} =\ 0)$$
$$(p_{k\,y}\ \leqslant\ c\ \wedge\ p_{k\,y} + s_{k\,y} > c\ \implies col_{k\,c} =\ s_{k\,x})\ \wedge\ (p_{k\,y}\ >\ c\ \vee\ p_{k\,y} + s_{k\,y} \leqslant c\ \implies col_{k\,c} =\ 0)$$

Finally we constrain the sum over each row and column to be equal or lower than the available paper size, width W and heigth H respectively:

$$\forall r \in W,\ \sum_{k}^{N} row_{k\,r} \leqslant H \qquad\qquad \forall c \in H,\ \sum_{k}^{N} col_{k\,c} \leqslant W$$

The presence of the *if-then-else* slows the propagation down, this is due to the *or* operator in the implication: to prove a disjunctive clause false the solver should falsify each and every of its literals.

We can thus improve the search by using a global contraint, which has better propagation.
The `cumulative` scheduling constraint is repurposed for our use case:
Originally, it "requires that a set of tasks given by start times s , durations d , and resource requirements r , never require more than a global resource bound b at any one time"

We use it on rows and columns by considering:

- integer bottom-left corner in place of start time,
- present x/y dimension in place of task duration,
- opposite y/x dimension in place of resource requirement,
- column/row lenght as global resource bound,

expressed in code for columns:

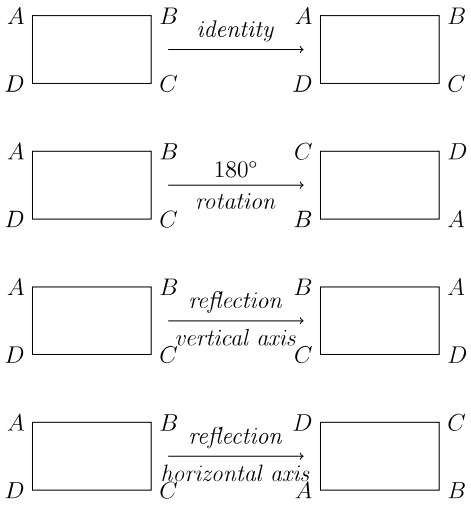
```
constraint cumulative(present_pos[1..n,1],
                      present_shape[1..n,1],
                      present_shape[1..n,2],
                      paper_shape[2]);
```

Where: `paper_shape[2]` is H
(respectively done for rows)

Marco Buiani
Daniele Verì

Dealing with symmetry

The present wrapping problem in its proposed form has a total of four symmetric solutions. They are given by identity, 180° rotation, horizontal reflection and vertical reflection.



To save the solver some work we could let it generate non-symmetryc solutions only. More importantly, the solver would explore symmetric states of non-solution ones during search, considerably slowing the reach of a feasible one.

To improve this, we employ static simmetry-breaking.
The first present is confined inside the bottom left quadrant of available paper in order to reduce the search space. Constraining one present only allows to not lose non-symmetryc solutions.

$$p_{1\,x} < (W - s_{1\,x}) / 2 \quad \text{breaking horizontal symmetry}$$

$$p_{1\,y} < (H - s_{1\,y}) / 2 \quad \text{breaking vertical symmetry}$$

To not lose efficiency, this constraint is paired with a value search heuristic like indomain_min that selects smallest value first.

Guiding the search with heuristics

Inspired by common sense reasoning we guide the search by having the solver allocate bigger presents first.
In a first fail fashion, bigger presents have the smallest domain, this way a bad partial assignment will be spotted earlier during search, thus abandoning the infeasible sub-tree with

backtracking.

In order to do so we used the **first_fail** heuristic on **variables**, while **indomain_min** heuristic on **value** fills the available space from the paper border, this minimizes the chance of leaving small empty spaces.

Moreover, the proposed instances show a great asymmetry in the present shapes.
This suggests to explore the search space by "y" dimension first: the presents are taller than wide, thus the domains are inherently more constrained on the y coordinate.
Problem instances inherent asymmetry is shown by comparing the two search heuristics:

```
search_ann_y = int_search(present_pos[1..n,2], first_fail, indomain_min);
search_ann_x = int_search(present_pos[1..n,1], first_fail, indomain_min);
```

Results are shown for the 27x27 instance and can rbe eproduced for the others provided.
Gecode 6.1.1 solver runs on a single thread i7 4th gen CPU at 2.3 GHz.

Giving priority to x dimension, `search_ann_x`:

%%%mzn-stat: solveTime=15.1953
%%%mzn-stat: solutions=1
%%%mzn-stat: propagations=6237422
%%%mzn-stat: nodes=1246909
%%%mzn-stat: failures=623441
Finished in 15s 349msec

While when prioritizing y, `search_ann_y`:

%%%mzn-stat: solveTime=8.26603
%%%mzn-stat: solutions=1
%%%mzn-stat: propagations=2945627
%%%mzn-stat: nodes=1192600
%%%mzn-stat: failures=596288
Finished in 8s 420msec

The solving time almost halves for y, this suggests that an infeasible assignment is reached with less propagations (more than half less), thus less time.
In order to be more general we adopt a mixed approach, giving precedence to y but leaving x heuristic as a second choiche.

```
solve :: search_ann_y :: search_ann_x satisfy;
```

Shape rotation variant

In this problems variant we allow rectangular presents to rotate by 90 degrees steps.
In particular any rectangle has just two possible rotation states, given by swapping its height and width, a further 90° rotation would just set the rectangle back to its original orientation.
This problem relaxation greatly impact the search space dimension, incresing it by 2^N.

To handle this we make use of a boolean variables array, rot, to hold the rotation state of each piece.
We then use $rotshape_{k\,x}, rotshape_{k\,y}$ in place of present shapes $s_{k\,x},\ s_{k\,y}$, as an interface to take into account the possible rotation of the presents:

$$(rot_k \implies rotshape_{k\,x} = s_{k\,y}) \ \wedge (\neg rot_k \implies rotshape_{k\,x} = s_{k\,x})$$

$$(rot_k \implies rotshape_{k\,y} = s_{k\,x}) \ \wedge (\neg rot_k \implies rotshape_{k\,y} = s_{k\,y})$$

We prove the correctness of this approach by creating a suitable instances `WxH_rot.txt`, that can be solved only if allowing presents to rotate.

Same shape symmetry

The search space is reduced by a factor rf dependent on the number of equal presents:

Let e_{num} be the number of equivalence classes.
Let c_i be the cardinality of the i-th equivalence class.

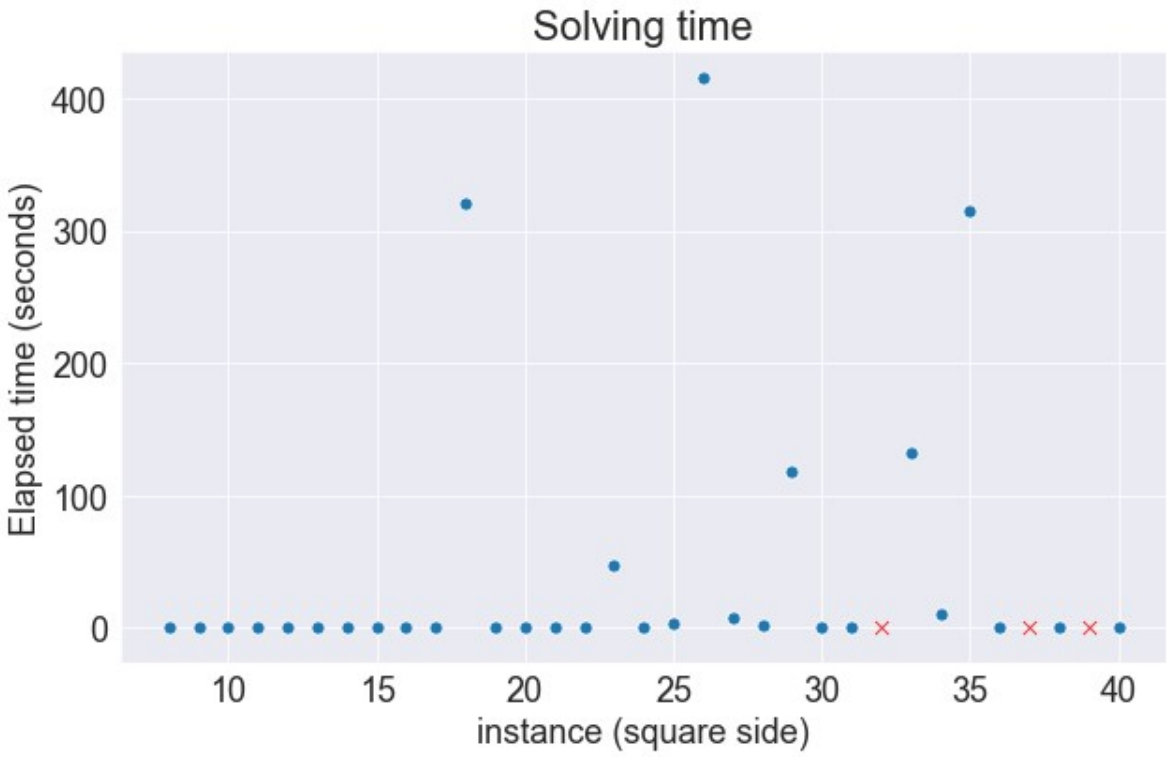
$$rf = \prod_{i=1}^{e_{num}} c_i!$$

To exploit this constraint further work is needed.
In particular, for every equivalence class of identical presents only one configuration should be explored removing the others, considered symmetric.

Marco Buiani
Daniele Verì

Results

Our Constraint Programming model is able to solve most of the proposed problem instances with ease. Plotted below are the solving times over instance side.



It is shown that most of the smaller instances take risible time, on these there are not more than 10^5 solver failures.
This lets us think that the model description and search heuristics do a great job in guiding the solver to a solution. The reason why these instances are easilly solved resides in them all having at least one present as tall as the available paper. First fail search heuristic guarantees that that present is assigned first (smaller domain variable).
The most difficult instances are the one composed of many, small pieces.

With regards to the rotations variant, a difference in solving time is seen, due to the increase in search variables.
While we manage to solve smaller modified instances fast, some of the bigger ones take considerably more time. This behavior is greatly related to the instance peculiarities.