Combinatorial Decision Making and Optimization

Present wrapping problem

CP solution

Filippo Orazi 0000928971
Andrea Rossolini 0000954735

February 2021

Contents

1 Introduction

The present wrapping problem is a combinatorial problem that focuses on fitting all the paper needed to wrap a certain amount of present in a bigger sheet of fixed dimension.
The instances we had to solve provided rectangular pieces to be cut out of a rectangular sheet, each piece and the sheet had a fixed dimension specified in the input files.

The present document explains the solution we studied to solve the proposed *Present Wrapping Problem*. The solution has been developed using **Constraint Programming** and the IDE **MiniZinc**.

2 Model

This section explains our model following the points listed in the project proposal.

2.1 Variables and Main Problem Constraints

All the variables are properly explained and commented on in the source code. Below the main variables are listed and discussed:

- **`n_pieces`** total number of pieces of paper.
- **`rectangles`** dimension of each piece of paper.
- **`coordinates`** coordinates of the left bottom corner.

The following bullet point includes the variables used into solve the "rotation problem":

- **`n_shapes`** number of rectangles' shapes (i.e. an $n \times n$ piece of paper is counted as 1 shape, while a $m \times n$ piece of paper as 2 shapes).
- **`rect_size`** the size of each box in 2 dimensions.
- **`rect_offset`** the offset of each box from the base position in 2 dimensions.
- **`shape`** the set of rectangles defining the i -th shape.
- **`shape_index`** index of shapes related to a paper and, eventually, its rotation.
- **`kind`** the shape used by each object.

Given the bigger sheet as the domain (D) with its dimensions (D_{height}, D_{width}) and the dimension of each piece p_i (p_{i_w}, p_{i_h}) we want to find the coordinates (p_{i_x}, p_{i_y}) of the lower left corner of each piece. The basic constraints for the main problem are:

- All the lower left corners need to stay in the domain:

$$\begin{aligned} \forall\, p \in P : 0 < p_y < D_{height} \\ \wedge \\ 0 < p_x < D_{width} \end{aligned} \tag{1}$$

- All the pieces need to fit in the domain:

$$\begin{aligned} \forall\, p \in P : 0 < p_y + p_h < D_{height} \\ \wedge \\ 0 < p_x + p_w < D_{width} \end{aligned} \tag{2}$$

- Two pieces can not overlap:

$$\begin{aligned} \forall\, p_i, p_j \in P \wedge i \neq j : p_{i_y} + p_{i_h} < p_{j_y} \\ \wedge \\ p_{j_y} + p_{j_h} < p_{i_y} \\ \wedge \\ p_{i_x} + p_{i_w} < p_{j_x} \\ \wedge \\ p_{j_x} + p_{j_w} < p_{i_x} \end{aligned} \tag{3}$$

2.2 Implied cumulative constraints

The implied constraints assure that the sum of the vertical dimension of all pieces in a certain column is not greater than the height of the domain (and similar for the width).

To satisfy this constraint, we used a predicate provided by MiniZinc standard libraries: `cumulative(s, d, r, b)`, where `s` is an array with all the paper's origins $P_{o,i}$ (with $i = x \vee i = y$), `d` and `r` are the arrays with the paper's dimensions (p_w, p_h), while `b` is the integer that represent the paper roll limit that bounds the papers, hence w or h.

2.3 Global Constraints

2.3.1 Non-overlapping constraint

To achieve this constraint we used a predicate available in the MiniZinc standard library. This predicate is `diffn(x, y, dx, dy)`[1], where `x` and `y` are the origin of the rectangle, while `dx` are `dy`. Using a formal definition we can write:

[1] https://www.minizinc.org/doc-2.5.3/en/lib-globals.html

$$\begin{aligned} &\forall\ p_i, p_j \in P: \\ &(p_{jo,x}, p_{jw}) \notin [p_{io,x}, p_{io,x} + p_{iw}] \\ &\wedge \\ &(p_{jo,y}, p_{jh}) \notin [p_{io,y}, p_{io,y} + p_{ih}] \\ &with\ i \neq j \end{aligned} \tag{4}$$

(other interesting predicate to study are `diffn_k` and `geost_nonoverlap_k`)

2.3.2 Avoiding symmetries

To avoid symmetrical solutions we added a constraint that forces the first block on the bottom left side of the paper. In this way, the number of possible solutions decreases. In addition, the search speeds up due to the less number of solution branches. This constraint has not always been used in our tests, considering that in some cases, forcing a block, invalidates all the solution.

2.4 Best way to search for solutions in CP

Using the searching annotation listed on MiniZinc documentation [2] we performed several experiments with different kind of solving behaviours. As a result, the search strategy that achieves the most results with the less number of failure is *First fail*.

2.5 Rotation

To deal with the rotation problem, we consider all the $n \times m, n \neq m$ rectangles (non-square) as two separate rectangles and let the constraints decide which one of the two rectangles is better to solve the problem. To reach the result, we used the MiniZinc built-in function `geost_bb(k, rect_size, rect_offset, shape, x, kind, l, u)` [3]. This global constraint ensures that the given k-dimensional objects do not overlap and that all pieces fit within a global k-dimensional bounding box.
In our solution the parameter `rect_size` contains all the rectangles and their rotation in case they are non-squared. The variable `rect_offset` represents all the starting point of the rectangles. Thus, in the beginning, it will be an array of couples $[[0,0],...]$. All the other variables are used as indicated in the MiniZinc documentation.

2.6 Same dimension pieces

To deal with this problem, we sort the pieces of the same size using their lexicographical order. To achieve this solution, we used a predicate from MiniZinc

[2] https://www.minizinc.org/doc-2.4.3/en/lib-annotations.html
[3] https://www.minizinc.org/doc-2.5.3/en/lib-globals.html#index-97

documentation: **lex_greater**(array [int] of var int: x, array [int] of var int: y). Thus, iterating all the rectangles we have for the problem if two or more rectangles have the same shape (this information is stored in the .dzn file), they got ordered by the predicate explained above.

3 Results

All the results have been computed using MiniZinc. Moreover, all the results not reported in the tables are the tests that take more than 1000 seconds to converge.

3.1 CP without rotation

The following table reports all the results computed by the CP solution that does not consider the pieces of paper rotation.

Paper size	time
8x8	785ms
9x9	974ms
10x10	786ms
11x11	809ms
12x12	840ms
13x13	797ms
14x14	810ms
15x15	817ms
16x16	793ms
17x17	774ms
18x18	-
19x19	576ms
20x20*	720ms
21x21*	560ms
22x22	581ms
23x23	1m 22s
24x24	5s
25x25	979ms
26x26*	2m 53s
27x27	6s
28x28	2s
29x29	2m 43s
30x30	579ms
31x31	583ms
32x32	-
33x33*	936ms
34x34	583ms
35x35	-
36x36	-
37x37	-
38x38	570ms
39x39	-
40x40	680ms

Table 1: CP results with no rotation
* results computed without forcing the first block (section 2.3.2)

3.2 CP with rotation

In the table below, the results stop at 14×14 because all the other experiments presented a computation time greater than 1000 seconds.

Paper size	**time**
8x8	680ms
9x9	765ms
10x10	785ms
11x11	1s
12x12	1s
13x13	18s
14x14	4m 9s

Table 2: CP results considering rotation