Project in Combinatorial Decision Making and Optimization - Present Wrapping Problem (PWP) - CP

Petru Potrimba
petru.potrimba@studio.unibo.it

December 7, 2020

Contents

1 Constraint Programming (CP)

In this section we are going to discuss the solution of the Present Wrapping Problem (PWP) using Constraint Programming (CP) with MiniZinc tool. We start from the data and then we describe the chosen variables as well as the constraints of the problem.

1.1 Data

The given data are in the following format:

```
8 8
4
3 3
3 5
5 3
5 5
```

where the two numbers in the first line represents the width and the height of the paper roll, the single number in the second line represents the number of piece of papers to cut and the other lines represent the width and height for each piece of paper we need to cut.

1.1.1 Data format for MiniZinc

In order to work with MiniZinc, we need to change the format of the data. To achieve that, I wrote a python script that takes as input the previous data format and outputs the following format:

```
width = 8;
height = 8;
blobs_count = 4;
blobs = [| 3, 3
         | 3, 5
         | 5, 3
         | 5, 5
|];
```

1.2 Parameters and variables

In this section we are going to illustrate the parameters and variables I chosen for the problem:

1.2.1 Parameters

```
# represents the paper width
int: width;
# represents the paper height
int: height;
# represents how many piece of papers is required to cut
int: blobs_count;
# array of `blobs_count` length containing the
# width and height per each piece of paper
set of int: BLOBS_ARRAY = 1..blobs_count;
array[BLOBS_ARRAY, 1..2] of int: blobs;
```

1.2.2 Variables

```
array[BLOBS_ARRAY, 1..2] of var int: xy_coord;
```

This variable is the array that will contain the solution of the problem. It is an array composed of pairs of `(x, y)` coordinates which represents the starting point of the respective piece of paper to cut in `blobs` array. At the end this array will be structured as follows: `[(x, y), (x, y), (x, y), ...]` where `x` represents the *width* while the `y` represents the *height* per each piece of paper.

1.3 Constraints

In this section we are going to describe constraints I designed for the problem, starting from the basic constraints till the more challenging ones.

1.3.1 Bounding variables

This constraint ensures that the domain of the `xy_coord` variable does not exceed the ranges `[0, width[` and `[0, height[`.

```
constraint forall(i in BLOBS_ARRAY)
                    ((xy_coord[i,1] >= 0 /\ xy_coord[i,1] < width) /\
                    (xy_coord[i,2] >= 0 /\ xy_coord[i,2] < height));
```

1.3.2 Paper roll fit

This constraint ensures that each piece must fit in the paper roll. In particular this constraint come from the sentence idea "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, a similar property holds if we draw a horizontal line".

```
constraint forall(i in BLOBS_ARRAY)
                  (((xy_coord[i,1] + xy_coord[i,1]) <= width) /\
                  ((xy_coord[i,2] + xy_coord[i,2]) <= height));
```

1.3.3 Non overlapping

The following constraint ensures that the pieces we need to cut do no overlap each other.

```
constraint forall(b1, b2 in BLOBS_ARRAY where b1 < b2) (
                    (xy_coord[b1,1] + blobs[b1,1] <= xy_coord[b2,1]) \/
                    (xy_coord[b2,1] + blobs[b2,1] <= xy_coord[b1,1]) \/
                    (xy_coord[b1,2] + blobs[b1,2] <= xy_coord[b2,2]) \/
                    (xy_coord[b2,2] + blobs[b2,2] <= xy_coord[b1,2]));
```

1.4 Global and implied constraints

It turns out that some constraints could be expressed using the Global Constraints that MiniZinc offers. This lead into a cleaner and more efficient constraint definition. In particular, the previous `Non overlapping` constraint can be defined in another way using the `diffn` Global constraint as follows:

```
constraint diffn(xy_coord[..,1], xy_coord[..,2], blobs[..,1], blobs[..,2]);
```

Also, the previous `Paper roll fit` constraint can be defined in another way using the `cumulative` Global constraint as follows:

```
constraint cumulative(xy_coord[..,1], blobs[..,1], blobs[..,2], height);
constraint cumulative(xy_coord[..,2], blobs[..,2], blobs[..,1], width);
```

This latter constraint is basically checking whether for each column in the paper roll the the sum of the heights of the pieces on that row is at most the paper roll height. Analogously for the rows.

1.5 Symmetry breaking

I applied also a symmetry breaking constraint that helped to reduce the number of failures. The idea has been taken from this MiniZinc tutorial [4] and it states that the first object to cut should be the in the bottom left side paper roll. This counteracts equal-rotated solutions.

```
constraint (xy_coord[1,1] <= (width/2 - blobs[1,1]/2)) /\
           (xy_coord[1,2] <= (height/2 - blobs[1,2] / 2));
```

Another interesting extension that could be added to the previous constraint is to order the pieces of paper to cut in a decreasing order (based on their area) and place the big one in the bottom left side of the paper roll. This should decrease the number of failures and thus speed up the search.

1.6 Rotation

In this section we are going to discuss how we can deal with the rotations of the pieces of papers to cut. Suppose the the case depicted by the below figure:

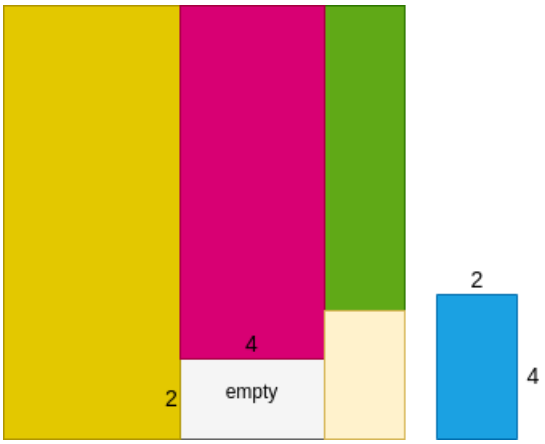

Figure 1: Rotation dealing.

this case would be a fail. However, we can easily notice that by rotating the blue piece by 90° degrees, this would lead into a successful solution. To implement this rotation case, we can think of having a double number of pieces to cut of and performing the search making sure that only the right number of piece should be cut from the paper roll. Namely, if we have a $n \times m$ piece to cut, we can think of adding another one of size $m \times n$. Of course, if the transpose of a piece is the piece itself, it does not make any sense in adding a new equal piece. Of course this would slow down our search since there is a new shape for each piece to take into account. This could be implemented in MiniZinc using the global constraint `geost_bb` [3] or `diffn_k` [2].

1.7 Equal pieces

If there can be multiple pieces of the same dimension, we would end up in having multiple symmetric solutions. We show an example below:

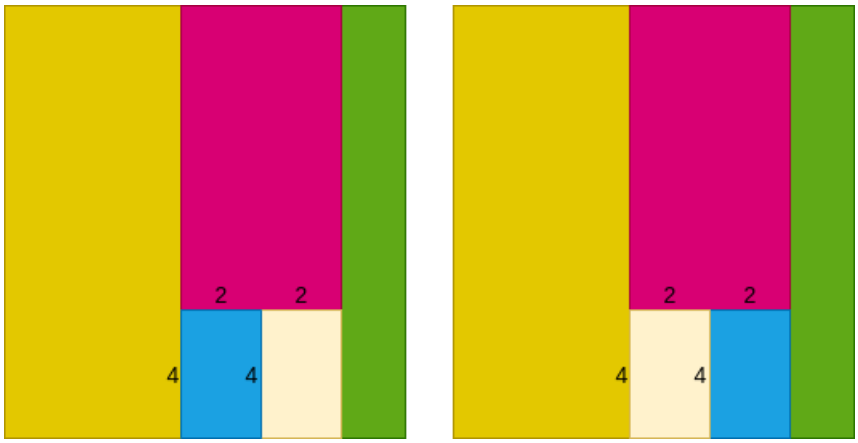

Figure 2: Equal pieces daeling.

As it can be seen, these are two equivalent solution. Thus, we need to avoid exploring them. To solve this problem, first of all is necessary to check if we actually have pieces of the same dimension. Done that, supposing that in our instance list we have two equal pieces, one of the two must arbitrarily be placed in a position x with respect to the other (for instance one must be placed in a position x and the other in the position $x+1$). In this way, with this constraint, we exclude regardless of everything that does not respect that constraint, that is, all the symmetrical solutions that regards equal pieces. This can be achieved using a `lex_lesseq` [1] lexicographic global constraint of MiniZinc.

1.8 Search strategies

I tested three different search strategies, in particular:

```
1- s1 = int_search(xy_coord, first_fail, indomain_split);
2- s2 = int_search(xy_coord, dom_w_deg, indomain_split);
3- s3 = int_search(xy_coord, most_constrained, indomain_split);
```

All the search strategies were able to solve some certain instances and were not able to solve some others. Also, I tested the these strategies with *indomain_min* instead of *indomain_split* but no evident differences in performance arised. In the end, the best search strategy in terms of time to find a solution and in terms of failures turns out to be the *first one*.

1.9 Results

In this section I outline the results I obtain for all the instances with a time limit of 3600 seconds:

Instance	N°failures	Seconds
8x8	0	0
9x9	0	0
10x10	0	0
11x11	5	0
12x12	34	0
13x13	0	0
14x14	40	0
15x15	168	0
16x16	3	0
17x17	2902	0
18x18	-	-
19x19	913927	10
20x20	162224818	1667
21x21	187023236	2047
22x22	27445	0
23x23	8422200	67
24x24	315323	2
25x25	76361	1
26x26	4990096	40
27x27	2439744	20
28x28	796338	7
29x29	6566456	49
30x30	36117	0
31x31	2	0
32x32	-	-
33x33	-	-
34x34	453	0
35x35	-	-
36x36	-	-
37x37	-	-
38x38	32	0
39x39	14564773	154
40x40	3	0

References

[1] The MiniZinc Handbook. *Lexicographic constraints.*

[2] The MiniZinc Handbook. *Packing constraints - diffn_k.*

[3] The MiniZinc Handbook. *Packing constraints - geost_bb.*

[4] The MiniZinc Handbook. *Static Symmetry Breaking.*