Alma Mater Studiorum
University of Bologna

Combinatorial Decision Making and Optimization

Report CP Model

Dicosola Alessandro [Matr. 935563]

Base model

The problem is encoded using as decision variables X and Y that are the bottom left coordinates of each rectangle thus $X[i]$ and $Y[i]$ represents the x and y of the i-th rectangle (called Present inside the model).

The constraints used are the following:

1. X and Y should be bounded inside the "available" area for them: $0 < x_i < W - w_i$ and $0 < y_i < H - h_i$ where W,H are the total width and height of the whole area and w_i and h_i are the width and height of the considered rectangle.

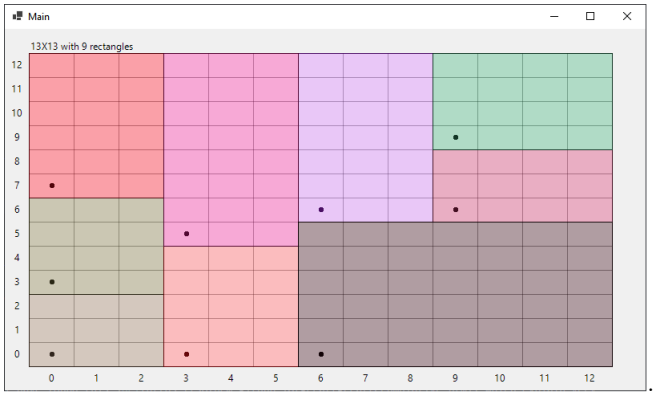
2. No rectangle has to overlap using **diffn**

3. The sum of the width (resp. height) of each rectangle within the same x (resp. y) has to be equal to H (resp. W) (although only the one that constraint X is used). At the beginning I wrote

   ```
   constraint forall(x in 0..W-1 where count(X,x)>0)
       (sum(p in Presents where X[p] = x)
           (heights[p])=H);
   ```

   which results to UNSATISFIABLE for some instances since we can see that misses some point (for $x = 9$ the sum doesn't consider the point $(6, 0)$) ; instead this works well

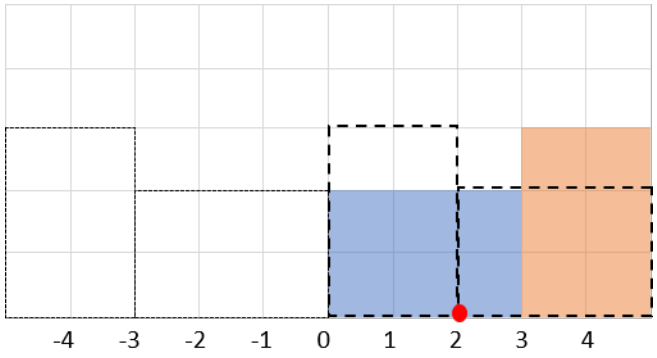
   ```
   constraint forall(x in 0..W-1 where count(X,x)>0)
       (sum(p in Presents where X[p] <= x /\ X[p]+widths[p]>x)
           (heights[p])=H);
   ```

4. In order to reduce the search space:

   - The x and y of the largest rectangle are forced at the beginning since it's the most difficut because can be inserted in few position instead a small rectngle it's esier and as consequence the domain of X is reduced.
   - A redundant global constraint is used: **comulative**. We can see:
     - the starting point as the coordinate x or y
     - the duration as the width or height
     - the resource requirements as height or width
     - the resource bound as H or W

     thus at each x (resp. y) of a rectangle the height (resp. width) should not exced the H (resp. W).
     NOTE. This was possibile thanks to
     `https://sofdem.github.io/gccat/gccat/Ccumulative.html` which allow me to see this possibility.



Flip over y

- Avoid symmetries.
  NOTE. The symmetries are permutation of the solution but in this case the symmetric solutions have different values thus I'm not sure that this constraints are good
  - Flip over y. Assuming that we have the following configuration (blue and orange rectnagle). It's possibile to see that $-(x + width) + W$ gives the new point: in this case $-(0 + 3) + 5 = 2$ (the red point)
  - Flip over x. The same but $-(y + height) + H$
  - Rotation. I have applied the *rotation matrix* to the point which will be the new bottom-left point after the translation around the center:
    $X - C_x = (X - C_x)cos(\theta) - (Y - C_y)sin(\theta)$
    $Y - C_y = (X - C_y)sin(\theta) - (Y - C_y)cos(\theta)$
    For instance in the model I have applied only a 90° rotation (towards left) to the top-left point (which is the bottom-left after the rotation) since I'm not sure about the correctness (for the problem presented at the beginning about the symmetries).
    But the same logic should be applied to the other rotations.
    The following results were found:

Instance	No symmetries breaking	+Flip over x and over y	+Rotation
13×13	220	50	18
15×15	2688	944	456

Solver

The solver used is **Chuffed** since allow the satisfability of harder problem than *Gecode* (probably caused by the fact of weak symmetries breaking constraint).

Instance	Gecode (6 Threads)	Chuffed
40x40	189ms	885ms
39x39		1s 372ms
37x37		2m 36s
27x27	46s 261ms	101ms

Chuffed solver has a strategy called **priority_search** [1] which allow to select a strategy among many based on the delgate x and the strategy select and explore.

```
annotation priority_search(array [int] of var int: x,
                           array [int] of ann: search,
                           ann: select,
                           ann: explore)
```

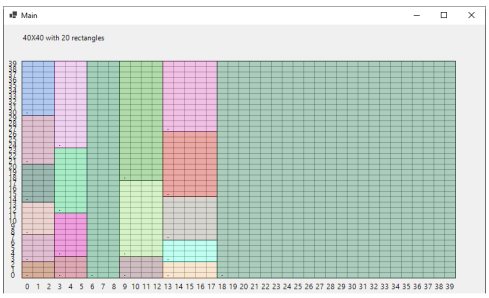
Using it in this way:

[1] https://www.minizinc.org/doc-2.4.3/en/lib-chuffed.html#functions-and-predicates

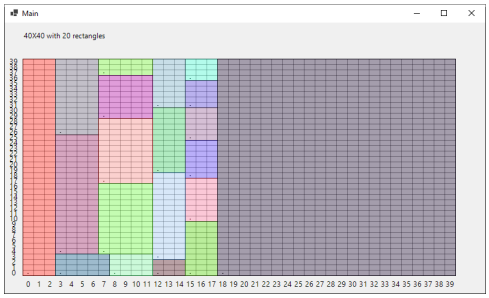
```
include "chuffed.mzn";
solve ::seq_search([
          priority_search(areas,
                    [int_search([X[p]],smallest,indomain_min) | p in Presents],
                    largest,indomain_max),
          int_search(Y,largest,indomain_max)
          ]) satisfy;
```

we associate each variable with its area and then we select the variable with the greatest area and assign to it the smallest value for X and largest for Y.

This behaviour can be see graphically:



(a) 40x40 using Gecode and seq_search



(b) 40x40 using Chuffed and priority_search

NOTE. The folder *out* contains the results found using **Chuffed** with **priority_search**

Model with rotation

Since we are using CP, we can take into account the rotation simply changing the *diffn* constraint with the global constraint **geost_bb**:

```
constraint geost_bb(2,RECT_SIZE,RECT_OFFSET,RECT_SHAPE,VARS,KIND,[0,0],[W,H]);
```

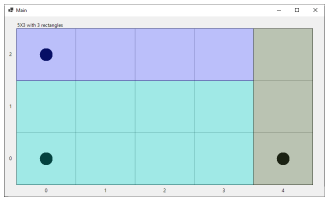
- RECT_SIZE contains the N × M and M × N sizes
- RECT_OFFSET contains the offset from the bottom-left point: in this case are all 0s since if we rotate the rectangle the base point is the same.
- RECT_SHAPE represent an "id" for each rectangle used:
- The set of SHAPES $1, 2, 3, 4, 5, ...$ is associated with the RECTS $w_1 \times h_1, h_1 \times w_1, ...$

In this case RECT_SHAPE simply define a rectangle with a particular width and height and an offset from the bottom-left point but in a more general problem the shape is a combination of rectangles that define a complex shape.

Figure 2: The red point is the offset (0,0) from the base point

I had to insert a channeling constraint for define VARS using X and Y and redefine the boundaries of X and Y and reimplement the implicit constraint and also the following constants:

With the toy problem present inside *model2.mzn* lead to the following solution (the last column is the shape used.):



```
5 3
3
4 2 0 0 1
4 1 0 2 3
3 1 4 0 6
```

Thus are used the rectangles:

- 4x2 with the shape 1 which is 4x2
- 4x1 with the shape 3 which is 4x1
- 3x1 with the shape 6 which is 1x3

Model same size

In this case imposing an ordering it's easier: we have to impose an ordering on both X and Y. The first time I wrote:

```
constraint forall(p,q in Presents where same_size(p,q))
    (lex_less([X[p],Y[p]],[X[q],Y[q]]));
```

which leads to UNSATISFIABLE thus I had to take into account the index and imposing the ordering using the index:

```
constraint forall(p in Presents)(
    let {array[int] of int: Q = [q | q in Presents where same_size(p,q)] } in
        forall(i in index_set(Q) where i > 1)
            (lex_less([X[i-1],Y[i-1]],[X[i],Y[i]])));
```

Using this toy problem

```
N=6;
W=7;
H=4;
widths = [2,2,2,2,2,1];
heights = [4,2,2,2,2,4];
```

Without constraint	With constraint
288 solutions	16 solutions

which are the following:

```
X:[0, 2, 2, 4, 4, 6] Y:[0, 0, 2, 0, 2, 0]
X:[0, 2, 2, 4, 4, 6] Y:[0, 0, 2, 2, 0, 0]
X:[0, 2, 2, 5, 5, 4] Y:[0, 0, 2, 0, 2, 0]
X:[0, 2, 2, 5, 5, 4] Y:[0, 0, 2, 2, 0, 0]
X:[0, 2, 4, 4, 2, 6] Y:[0, 0, 0, 2, 2, 0]
X:[0, 2, 4, 4, 2, 6] Y:[0, 2, 0, 2, 0, 0]
X:[0, 2, 5, 5, 2, 4] Y:[0, 0, 0, 2, 2, 0]
X:[0, 2, 5, 5, 2, 4] Y:[0, 2, 0, 2, 0, 0]
X:[0, 3, 3, 5, 5, 2] Y:[0, 0, 2, 0, 2, 0]
X:[0, 3, 3, 5, 5, 2] Y:[0, 0, 2, 2, 0, 0]
X:[0, 3, 5, 5, 3, 2] Y:[0, 0, 0, 2, 2, 0]
X:[0, 3, 5, 5, 3, 2] Y:[0, 2, 0, 2, 0, 0]
X:[1, 3, 3, 5, 5, 0] Y:[0, 0, 2, 0, 2, 0]
X:[1, 3, 3, 5, 5, 0] Y:[0, 0, 2, 2, 0, 0]
X:[1, 3, 5, 5, 3, 0] Y:[0, 0, 0, 2, 2, 0]
X:[1, 3, 5, 5, 3, 0] Y:[0, 2, 0, 2, 0, 0]
```