CPS - Constraint programming

Gonzalo Solera

May 2020

1 Optimal solution bounds

Before specifying the model for a given instance, I do a simple preprocessing of the instance to help to speedup the solution search. This step consists in finding an upper bound L' and a lower bound l' of the optimal solution L^*.

I define L' to be the roll length used when the boxes are naively stacked vertically. I define l' as $l' = \left\lceil \frac{\sum_i x_i y_i}{W} \right\rceil$, since this is the optimal length used if we were able to arbitrarily modify the shape of the boxes.

The bounds are not tight at all for arbitrary instances, but for the type of instances given in this project, the lower bound is quite similar to the optimal solution in most of the cases. This helps a lot to determine that the optimal solution has been reached.

2 Model

I have modeled the problem with 2 sets of N variables, using 2 variables for each box. One variable, namely p_i (pos) determines the position of the box i and another variable named d_i (dir) specifies the orientation of box i.

Initially, I used a pair of variables p_i^x and p_i^y to describe the position of a box, but the number of variables were considerably bigger and it was difficult to use a branching heuristic. Having a single position variable is possible by assigning a number to each cell of the paper roll as follows: $c_{xy} = Wy + x$. This allows a direct translation from a cell number to x-y coordinates by: $x = c_{xy} \mod W$ and $y = \lfloor c_{xy}/W \rfloor$.

To model the non-overlapping constraint of the boxes, I have used 4 auxiliary boolean variables for each distinct pair of boxes. I needed them to be able to express a disjunction of 4 constraints. I am not aware of any mechanism offered by gecode to solve this problem, but I didn't look much into it.

3 Optimizations

I have applied some minor optimizations that don't deserve to be mentioned here. The most important optimizations are related to the branch heuristic:

- I wanted to prioritize the allocation of the boxes in the top-left positions. This way, the boxes are not randomly placed across the paper roll, avoiding leaving empty small spaces between them that are impossible to fill afterwards. This is naturally enforced by prioritizing small values of p_i thanks to the previously mentioned codification of the position variables.
- I wanted to prioritize the allocation of the biggest boxes first, since they are the most difficult ones to place. For that, I use a custom and simple merit function that takes into account the area of the boxes.
- I wanted to enforce the boxes to be rotated if possible, since $y \geq x$. This is naturally accomplished by prioritizing small values of d_i, since $d_i = 0$ means that the box i is rotated.

4 Results

My proposed model of the problem offers good results. From the 108 given instances, it successfully finds the optimal solution of 101 instances within 60 seconds each. In the 7 other instances it finds a valid solution but it doesn't terminate, which means that the model can't confirm that the found solution is optimal. However, among these 7 instances where the model doesn't commit to a solution, in at least 5 instances the found solution is actually optimal.