Combinatorial and Decision Making Project Report (CP model)

Alberico Arcangelo

July 2020

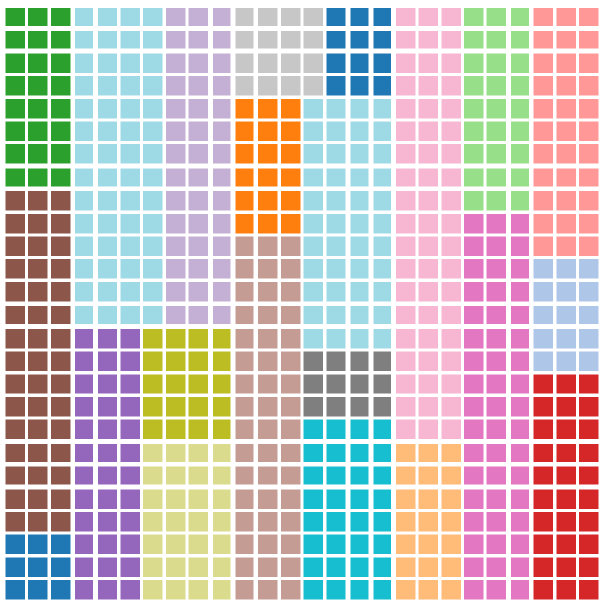

Figura 1: A solution for 26x26 instance

1 Model

The problem of PWP can be rephrased in this way: given a strip of width W and height H, given a set of rectangles $R_1, \dots, R_n$ with widths w_i and heights h_i, find a way (if exists) to pack these rectangles in the strip without overlapping the rectangles. In case of CP I focused on two different models: the first basic to solve the easier instances and a more clever one, in my humble opinion, in order to deal with more complex instances.
In the **basic model** the variables of the problem are the coordinates of the left-bottom corner of each rectangle in the strip.
After declaring the variables, we can start to write the constraint of our problem. I focused on three classes of constrains: domain, non-overlapping and implied.
Regarding the **domain constraints**, we have to take into account the fact that the rectangles must stay in the strip. So, given the pair of coordinate of the left-bottom corner (x_i, y_i) of the rectangle R_i, we have that:

$$0 \leq x_i \leq W - w_i$$

$$0 \leq y_i \leq H - h_i$$

For the **non-overlapping constraints** I considered the global constraint `diffn(x, y, w, h)` which constrains rectangles R_i, given by their origins (x_i, y_i) and sizes w_i and h_i, to be non-overlapping.
Concerning the **implied constraints**, if we draw an horizontal or a vertical line, the sum of the traversed pieces can be at most W in the horizontal case and H in the vertical one. So, we have to impose that for each $0 \leq i \leq h$:

$$\sum_{\substack{j=0 \\ (y_j \leq i) \wedge (y_j + h_j > i)}}^{n} w_j \leq W$$

for the horizontal case and that for each $0 \leq i \leq w$:

$$\sum_{\substack{j=0 \\ (x_j \leq i) \wedge (x_j + w_j > i)}}^{n} h_j \leq H$$

for the vertical case.
In the other model, called **multi model**, the variables are always the coordinates of the left-bottom corner but I splitted the problem into two easier ones: the first to find the coordinate x and the other to find the coordinate y

of the rectangles. I treated the first problem as a scheduling problem because I thought that each rectangle R_i could be regarded as a task with a certain duration w_i and a certain amount of resource usage h_i. So the first problem is reduced to find the starting time x of each task which can be done with a `cumulative` global constraint. Then, after finding the variables x, we are almost done because the finding of y is easier given that most of the complexity lies in the scheduling problem. So, in order to complete the problem I used a `diffn` global constraint with the previously found x.

2 Results

All the instances have been successfully solved in one of the two models. Among the solved some of them could be easily solved without specifying any particular search strategy. These instances are reported in the following table with the respective execution time and number of failures.

Instance	Time (seconds)	Failures
8	0.004	0
9	0.001	0
10	0.001	0
11	0.002	2
12	0.013	154
13	0.004	23
14	0.010	60
15	0.002	0
16	0.003	3
17	0.032	144
19	40.567	198617
20	8.068	33726
21	114.295	383858
22	94.799	342065

All the other instances were not solvable in a reasonable amount of time with the first model. Trying with different search strategies and restarts I manage to have better results for some models, for example, for instances 29 and 38, I applied an `int_search` with `dom_w_deg` as variable choice, `indomain_random` as constraint choice and `restart_luby(100)` as restart getting these results:

Instance	Time (seconds)	Failures	Restarts
29	350.091	811559	1743
38	598.421	1264367	2427

but again most of the remaining instances were not solvable in a reasonable amount of time. So, I tried the second model getting very good results, which I provide in the following table for each instance. Given that this time the problems to be solved are two, the results are given by the sum of both.

Instance	Time (seconds)	Failures
18	0.004	0
23	0.074	890
24	0.003	0
25	0.003	1
26	0.005	1
27	0.003	0
28	0.005	6
29	0.004	0
30	0.004	1
31	0.003	16
32	0.005	0
33	1.619	14862
34	0.627	7269
35	0.009	19
36	0.006	2
37	2202.100	13716941
38	224.629	1466412
39	0.011	31
40	0.007	3

Here we can see that even the most difficult instances were solvable in a matter of seconds, except for the instance 37 which exceeded the 10 minutes of running time. For this instance I had to play a little bit with search strategies and restarts in order to have this result. The best combination found was an `int_search` with `first_fail` as variable choice and `indomain_min` as constraint choice, while all the others strategies, even with restarts, led to more than 1 hour of running.

3 Generalization

The generalized problem provides the rotation of the rectangles and the handling of pieces of same dimension.
In order to deal with rotation I added a boolean array of variables which indicates if an element is rotated or not and then other two arrays of variables representing the actual width and height modified according to the value of

the rotation array. For this I found the CP model slightly more difficult than the SAT/SMT to be modified in order to account the rotation.
Then, regarding rectangles with the same dimension, I forced a relation between the two positions, that is to say if there are two rectangles R_i, R_j with $w_i = w_j$ and $h_i = h_j$ then:

$$x_j \geq x_i + w_i \vee y_j \geq y_i + h_i$$

To check the correctness of this changes, I provided two modified instances in the same folder: one is an 11x11 instance which would be infeasible without rotation and the other is a 12x12 instance with many pieces of equal dimension.