Combinatorial Problem Solving - BoxWrapping Constraint Programming with Gecode

Sergio Mosquera Dopico

May 22, 2020

1 Introduction

We think of it as a floorplanning problem, in which we will place the different boxes trying to minimize the total area as much as possible. The difference between this problem and a usual floorplanning one is that we have fixed the width value on the table, and what we want to minimize is just the value of the length.

2 Variables

For sake of simplicity, in further sections we will use the following nomenclature for the variables:

- B is the number of boxes read from the input file
- The width and length of box i, will be called w_i and l_i, respectively.
- The x and y coordinates for box i, will be called x_i and y_i, respectively.
- z_i will denote the value of the rotation for box i.
- rel_{ij}^x and rel_{ij}^y represent the relative position with respect to x and y, respectively for two boxes i and j.

2.1 Decision Variables

1. **IntVarArray**[] *x_coordinates* and *y_coordinates*.

   Will hold the position of the left upper corner of the box. The size for both arrays is the same, the number of boxes to be placed, but their values' range is different. While for *x_coordinates* we use values from 0 to $paper\_width - 1$, with *y_coordinates*, as we do not know before hand the size of this dimension, we will use values between 0 and the maximum possible value for the length, that is the value M which is defined in Section 2.2

2. **BoolVarArray** *rotation*.

   Having the same size as B, holds boolean values to indicate whether the box at index i is rotated (1) or not (0), i.e. swap values of width and length.

3. **BoolVarArray** $rel^x_{i,j}$ and $rel^y_{i,j}$.

   Boolean arrays stating the relative position relationship between every pair of boxes. In Floorplanning the relative position of some pair of boxes is given by two unary values, that is:

   - $(0,0) \rightarrow$ box i is at the left of box j.
   - $(0,1) \rightarrow$ box i is below box j.
   - $(1,0) \rightarrow$ box i is at the right of box j.
   - $(1,1) \rightarrow$ box i is above box j.

   We can think on the relative position between every two boxes with a matrix representation, such that every box will establish a relationship with the boxes having a greater index. There will be no values in and below the main diagonal, so we are interested just in the values above this diagonal. For an instance with four boxes, we would have something like this:

$$\begin{pmatrix} - & x_{12} & x_{13} & x_{14} \\ - & - & x_{23} & x_{24} \\ - & - & - & x_{34} \\ - & - & - & - \end{pmatrix} \qquad \begin{pmatrix} - & y_{12} & y_{13} & y_{14} \\ - & - & y_{23} & y_{24} \\ - & - & - & y_{34} \\ - & - & - & - \end{pmatrix}$$

   The size of both variables is given by $\sum_{i=1}^{B} i$ and the value of the relative positioning for boxes i and j is given by the following equation (works for both x and y relatives):

$$rel_{ij}\left[i * B + j - \sum_{k=1}^{i+1} k \right] \tag{1}$$

4. **IntVar** *paper_length*.

   The variable holding the value to be minimized.

2.2 Non-Gecode Variables

Here lie all the information that is either parsed directly from the input file (as the boxes), or instead the one computed from this input

1. **BoxType**[] *boxes*.

   An array with the abstraction of the boxes parsed from file. Every element in the array stores the attributes for each box. It is not storing the number of boxes of each type, so we are not storing as many boxes as types of them, but the total boxes defined by the input file.

2. **int**[] *number_of_boxes*.

   Auxiliary array to control the number of boxes for each type. Helpful when accessing the boxes by index to enforce the constraints.

3. **int** *paper_width* and M.

   Integer variables holding the fixed value for the length of the paper and the maximum length that can be achieved by the worse placement of the boxes, respectively:

$$M = \sum_{i}^{B} max(w_i, l_i) \tag{2}$$

3 Constraints

3.1 Collisions

To avoid that two or more boxes collide, there are a set of constraints which take advantage of the relative position between the different boxes. As explained in section 2.1 there may be four different situations given the positioning of two boxes.

Each of the following equations models a single situation, such that the boxes involved in this equation respect the bounds of each other. The composition of these four equations avoids the collision among every pair of boxes in the solution.

$$x_i + z_i l_i + (1 - z_i) w_i \leq x_j + M(rel^x_{i,j} + rel^y_{i,j}) \tag{3}$$
$$x_i - z_j l_j + (1 - z_j) w_j \geq x_j + M(1 - rel^x_{i,j} + rel^y_{i,j}) \tag{4}$$
$$y_i + z_i w_i + (1 - z_i) l_i \leq y_j + M(1 + rel^x_{i,j} - rel^y_{i,j}) \tag{5}$$
$$y_i - z_j w_j - (1 - z_j) l_j \geq y_j + M(2 - rel^x_{i,j} - rel^y_{i,j}) \tag{6}$$

3.2 Paper bounds

In a common floorplanning problem we want to guess both the width and length of the minimum plane we need for placing all the blocks. These two values are obtained imposing two constraints In this case, we just need to guess the length as we already know the fixed width.

Anyway, we need to ensure that no blocks exceed the maximum width and at the same time that we minimize the length, so we will enforce two constraints as well in this case. We need to know the bounds of each box to impose the constraints, which can be obtained from the position of the box, its length and width, and if the box is rotated or not:

$$paper\_width \geq x_i + (z_i l_i + (1 - z_i) w_i) \tag{7}$$

$$paper\_length \geq y_i + (z_i w_i + (1 - z_i) l_i) \tag{8}$$

Notice the bounds are different depending on the dimension we want to check.

Last equation, is the one we will use to minimize the solution as it represents the value we are interested in guessing. There is no need to impose any constraint to check the bounds in the opposite directions, because we are using the upper left corner as the reference for the coordinates.

3.3 Pruning constraints

After the problem is modeled with all the necessary constraints, we explored the possibility of finding a solution faster by the addition of more constraints that help in pruning the search space. The constraints that were added, avoid more collisions which are not detected at first and reduce a lot the search space (some instances go from 1 minute to find a suboptimal solution to 1 second). This constraint just checks that two boxes do not share the same upper left corner. For two boxes i and j, we need to ensure that if they share some coordinate (either x or y) they do not share the other, too.

$$x_i = x_j \rightarrow y_i \neq y_j \tag{9}$$

$$y_i = y_j \rightarrow x_i \neq x_j \tag{10}$$

4 Search strategy

As we are dealing with an optimization (minimization) problem, the best option to be chosen for search is a Branch and Bound strategy, which keeps looking for a better solution at every step instead of returning all of the possible solutions as a BFS or DFS engine would do.

5 Results

The program is able to find a solution for around 50 instances within the time limit (one minute), but only for 32 of them it finds the optimal solution. Despite the efforts in trying to minimize the search space no significant changes in the results were found. For instance, I tried changing from an initial solution using a single array of coordinates for both x and y, by two independent arrays, such that tighter bounds can be imposed to each of them. Or even the inclusion of additional constraints to avoid more collisions.