CPS - Assignment 1

CONSTRAINT PROGRAMMING

May 6, 2018

Kymry Burwell
UPC

Contents

1 VARIABLE DEFINITION

For this box wrapping constraint problem, I decided to model all of the boxes as IntVarArrays that hold the x and y coordinates of the boxes. There are 6 variable arrays in total, listed below. The length of each array is equal to the number of boxes and each element of the array represents a single box. As a side note, when I used different combinations of normal c++ vectors, IntVarArgs, and IntVarArrays, instead of all 6 being IntVarArrays, the program seemed to run at about the same speed, so I went with 6 IntVarArrays, as I thought this would give me the most flexibility to optimize and branch.

- *IntVarArray xtl* - Array that represents the top left x-coordinate of all boxes.
- *IntVarArray ytl* - Array that represents the top left y-coordinate of all boxes.
- *IntVarArray xbr* - Array that represents the bottom right x-coordinate of all boxes.
- *IntVarArray ybr* - Array that represents the bottom right y-coordinate of all boxes.
- *IntVarArray xw* - Array that represents the width of the boxes. Allows the box to be placed horizontally or vertically.
- *IntVarArray yh* - Array that represents the height (length) of the boxes. Allows the box to be placed horizontally or vertically.

*Note - I use length and height interchangeably.

2 BASIC CONSTRAINTS

I decided on the following constraints, propagation, and branching strategies after trying numerous different methods, which I will briefly touch on throughout the following sections. Each of the constraints are numbered according to the order they appear in the source file, allowing for easy reference.

1. **Boxes can be placed horizontally or vertically.**
   This first constraint allows the width and height to be switched. For each box, there are two *dom()* and two *rel()* constraints. These are used with the *xw*(box width) and *yh*(box height) variable arrays and constrains their domains to two values: (1) height of the box and (2) width of the box. It also sets a relation between the two, ensuring that if one uses the height, the other must use the width, and vice versa. For example, If a box is 3x1, the

constraint enforces (width=1 –> height=3) and (width=3 –> height=1). As a bi-product, this constraint ensures squares are not flipped, as this would add no value to solving the problem, because they are squares.

2. **Boxes cannot overlap (uses bounds propagation).**
   Ensures that none of the boxes overlap. This constraint is a primary reason we need the *xw* and *yh* variables. It utilizes the *nooverlap()* constraint provided by Gecode, which takes 6 variable arrays as input. This same constraint could be written manually using multiple *rel()* constraints among the x and y-coordinates and the box dimensions.
   For this constraint, I tried multiple different propagation methods such as value propagation with IPL_VAL, domain propagation with IPL_DOM, as well as others. In the end, I found that bounds propagation that aims to achieve bounds consistency with IPL_BND was the most efficient overall.

3. **Boxes are the correct size**
   This constraint is an auxiliary constraint that is needed for the *nooverlap()* constraint above. It simply ensures that the box is the correct size. It constrains the bottom right coordinates to be exactly equal to the top right coordinates plus the width/height of the given box. That is, $xtl + xw == xbr$ and $ytl + yh == ybr$ for all boxes.

4. **Dimensions of the paper roll are not exceeded.**
   This constraint uses two *rel()* constraints that ensure the width and the height (length) of the roll are not exceeded. Although height isn't strictly necessary, because it should be dealt with in the branching strategy, I opted to use it in hopes that it would help optimize the problem a bit.

3 Branching

I have four branchers, one for each of the x and y coordinate variable arrays as follows:

- *xtl brancher*: For variable selection, uses tiebreak to select the variable with the largest degree and smallest domain. For value selection, uses a random variable. I chose to use random value selection because I didn't want all top-left x-coordinates to choose the min or max value, as this would stack the boxes, which would increase the roll length.

- *ytl brancher*: For variable selection, uses tiebreak to select the variable with the largest degree and smallest domain. For value selection, uses the minimum value, as we are trying to decrease the length of the roll.

- *xbr brancher*: Uses the same strategy as the *xtl brancher*.
- *ytl brancher*: Uses the same strategy as the *ytl brancher*.

4 Constrain Function

I added a constrain function that ensures any new solution will be better than the previous one, i.e., it only returns improved solutions. This uses the Branch and Bound search algorithm. The function works by first finding the largest bottom-right y coordinate of the previous solution. This is the total length of the paper roll so far. It then creates a set of *rel()* constraints that ensure all bottom-right y coordinates of the new solution are strictly less than the max of the previous solution. This is an important constraint because we don't care about finding new solutions that are equal to, or worse than the current solution.

5 Symmetry Breaking

There is one symmetry breaking constraint that states all boxes of the same size are interchangeable. Meaning, if a box has already been tried in a certain position, an identically sized box should not be tried in that same position, as the end result will be the same. In order to break this symmetry, I created multiple *rel()* constraints among identically size boxes that state each box of the same size must have a top left x/y coordinate that is increasingly larger. For example, if there are 3 boxes that are the same size, box1, box2, box3, the following constraint will be posted:

$$box1 \quad topleft\{x,y\} \quad < \quad box2 \quad topleft\{x,y\} \quad < \quad box3 \quad topleft\{x,y\} \tag{1}$$

So, if box1 had $topleft\{0,0\}$, then box two can be either $topleft\{1,0\}$ or $topleft\{0,1\}$ or larger.