Project in Combinatorial Decision Making and Optimization - Present Wrapping Problem (PWP) - SMT

Petru Potrimba
petru.potrimba@studio.unibo.it

December 7, 2020

Contents

1 Satisfiability Modulo Theories (SMT)

In this section we are going to discuss the solution of the Present Wrapping Problem (PWP) using Satisfiability Modulo Theories (SMT) with the Z3 library of Python.

1.1 Data

The given data are given in the same format as I highlighted in the CP part and, contrary as in this case we do not need to do extra processing.

1.2 Parameters and variables

In this section we are going to illustrate the parameters and variables I chosen for the problem:

1.2.1 Parameters

```
# represents the paper width
width: integer
# represents the paper height
height: integer
# represents how many piece of papers is required to cut
blobs_count: integer
# array of `blobs_count` length containing the
# width and height per each piece of paper
blobs: list
```

1.2.2 Variables

```
xy_coord = [[ Int("o_{}_{}".format(i+1, j+1))
        for j in range(2)] for i in range(blobs_count)]
```

This variable is the array that will contain the solution of the problem. It is an array composed of pairs of `(x, y)` coordinates which represents the starting point of the respective piece of paper to cut in `blobs` array. At the end this array will be structured as follows: `[(x, y), (x, y), (x, y), ...]` where `x` represents the *width* while the `y` represents the *height* per each piece of paper. It is the very same `xy_coord` variable used in CP.

1.3 Constraints

In this section we are going to describe the constraints I designed for the problem.

1.3.1 Bounding variables

This constraint ensures that the domain of the `xy_coord` variable does not exceed the ranges `[0, width[` and `[0, height[`.

```
domain = [ And(xy_coord[i][0] >= 0, xy_coord[i][0] < width,
                  xy_coord[i][1] >= 0, xy_coord[i][1] < height)
             for i in range(blobs_count)]
```

We can express this constraint in a mathematical way as follows:

$$\bigwedge_{i=0}^{blob\_count} (xy\_coord(i,1) \geq 0 \wedge xy\_coord(i,1) \leq width \wedge xy\_coord(i,2) \geq 0 \wedge xy\_coord(i,2) \leq height) \tag{1}$$

1.3.2 Paper roll fit

This constraint ensures that each piece must fit in the paper roll.

```
fit_paper = [ And(blobs[i][0] + xy_coord[i][0] <= width,
                  blobs[i][1] + xy_coord[i][1] <= height)
             for i in range(blobs_count)]
```

We can express this constraint in a mathematical way as follows:

$$\bigwedge_{i=0}^{blob\_count} (blobs(i,1) + xy\_coord(i,1) \leq width \wedge blobs(i,2) + xy\_coord(i,2) \leq height) \tag{2}$$

1.3.3 Non overlapping

The following constraint ensures that the pieces we need to cut do no overlap each other.

```
over = []
for i in range(blobs_count):
 for j in range(blobs_count):
  if (i<j):
    over.append(Or(xy_coord[i][0] + blobs[i][0]  <= xy_coord[j][0],
                           xy_coord[j][0] + blobs[j][0] <= xy_coord[i][0],
                           xy_coord[i][1] + blobs[i][1] <= xy_coord[j][1],
                           xy_coord[j][1] + blobs[j][1] <= xy_coord[i][1]))
```

We can express this constraint in a mathematical way as follows:

$$\bigwedge_{j=0}^{blob\_count} \bigwedge_{i=0}^{j}$$

$(xy\_coord(i,1) + blobs(i,1) \leq xy\_coord(j,1) \vee$
$xy\_coord(j,1) + blobs(j,1) \leq xy\_coord(i,1) \vee$
$xy\_coord(i,2) + blobs(i,2) \leq xy\_coord(j,2) \vee$
$xy\_coord(j,2) + blobs(j,2) \leq xy\_coord(i,2))$

1.4 Implied constraints

Now we are going to define the same implied constraints we applied also in CP.

```
imp = []
for i in range(width):
    for j in range(blobs_count):
        imp.append(Sum([If(And(xy_coord[j][0] <= i,
        i < xy_coord[j][0] + blobs[j][0]), blobs[j][1], 0)
        for j in range(blobs_count)]) <= height)

for i in range(height):
    for j in range(blobs_count):
        imp.append(Sum([If(And(xy_coord[j][1] <= i,
        i < xy_coord[j][1] + blobs[j][1]), blobs[j][0], 0)
```

This constraint is basically checking whether for each column in the paper roll the the sum of the heights of the pieces on that row is at most the paper roll height. Analogously for the rows.

1.5 Symmetry breaking

I applied also a symmetry breaking constraint that helped to reduce the number of failures. This idea is the same I adopted from the CP symmetry breaking part and it states that the first object to cut should be the in the bottom left side paper roll to counteract the equal-rotated solutions.

```
symmetry_b = [ And(xy_coord[0][0] <= (width/2 - blobs[0][0]/2),
           xy_coord[0][1] <= (height/2 - blobs[0][1]/2))]
```

We can express this constraint in a mathematical way as follows:

$$xy\_coord(1,1) \leq (width/2 - blobs(1,1)/2) \wedge xy\_coord(1,2) \leq (height/2 - blobs(2,2)/2) \tag{3}$$

1.6 Results

In this section I outline the results I obtain for all the instances:

Instance	N°Propagation	Seconds
8x8	8	0
9x9	13	0
10x10	226	0
11x11	133	0
12x12	3881	0
13x13	6254	0
14x14	26679	0
15x15	136	0
16x16	14585	0
17x17	31053	0
18x18	1867522	10
19x19	145533	0
20x20	136179	1
21x21	175841	1
22x22	447149	2
23x23	2218346	12
24x24	181773	1
25x25	447187	2
26x26	5868839	33
27x27	2255146	13
28x28	4050540	27
29x29	6319747	50
30x30	3505554	26
31x31	2663424	17
32x32	9308334	88
33x33	4000298	34
34x34	465856	3
35x35	2244131	16
36x36	3426496	26
37x37	-	-
38x38	354871	2
39x39	-	-
40x40	569839	3