SMT Model - Present Wrapping Problem

Filippo Lo Bue

September 2020

1 Description of the Problem

Given a wrapping paper roll of a certain dimension and a list of presents, decide how to cut off pieces of paper so that all the presents can be wrapped. Consider that each present is described by the dimensions of the piece of paper needed to wrap it. Moreover, each necessary piece of paper cannot be rotated when cutting off, to respect the direction of the patterns in the paper.

The purpose of this project is to solve PWP proceeding as follows:

1. Start with the variables and the main problem constraints.
2. In any solution, if we draw a vertical line and sum the vertical sides of the traversed pieces, the sum can be at most l.
3. Use global constraints to impose the main problem constraints and the implied constraints in your SMT model.
4. Investigate the best way to search for solutions in SMT.
5. If rotation was enabled? which of the SMT model encoding is easier to modify to take this into account? How would you modify that model/encoding?
6. there can be multiple pieces of the same dimension: how would you improve the SMT model encoding?

Points **1 - 4** are part of the Z3py program.
Point **5** is expressed using mathematical notation and it was included in a second Z3py program.
Point **6** is expressed using mathematical notation and it was included in a third Z3py program.

2 1^{st}model - create_model (Points 1-4)

2.1 Parameters

- **pr_w**: width of the wrapping paper roll.
- **pr_h**: height of the wrapping paper roll.
- **n_pieces**: number of the presents/pieces.
- **L**: dimensions of the piece of paper needed to wrap each presents.
- **index_largest_p**: index of the biggest present/piece by area.

2.2 Variables

Bottom left corner of each pieces

$$\begin{aligned}\mathbf{q} &= [[piece\_1\_x, piece\_1\_y] \\ &= [piece\_2\_x, piece\_2\_y] \\ &\ \ ... \\ &= [piece\_n\_x, piece\_n\_y]]\end{aligned}$$

$\mathbf{e}, \mathbf{f} \in \mathbb{Z}$

$\forall\, i, j,\ i < j \quad lr_{i,j} \in [false, true]$	$true$ if r_i are placed at the left to the r_j, otherwise $false$
$\forall\, i, j,\ i < j \quad ud_{i,j} \in [false, true]$	$true$ if r_i are placed at the downward to the r_j, otherwise $false$
$\forall\, i, \quad px_{i,e} \in [false, true]$	$true$ if r_i are placed at less than or equal to e, otherwise $false$
$\forall\, i, \quad py_{i,e} \in [false, true]$	$true$ if r_i are placed at less than or equal to f, otherwise $false$

2.3 Constraints

2.3.1 Domain

$\forall\, i, \ \ 0 \leq q[i]\_x \leq pr\_w - i\_width \ \ \& \ \ 0 \leq q[i]\_y \leq pr\_h - i\_height$

2.3.2 Symmetry breaking rules - largest rectangle domain reduction

$$0 \leq q[index\_largest\_p]\_x \leq \lfloor \frac{pr\_w - index\_largest\_p\_w}{2} \rfloor,$$

$$0 \leq q[index\_largest\_p]\_y \leq \lfloor \frac{pr\_h - index\_largest\_p\_h}{2} \rfloor$$

2.3.3 Order Encoding

$\forall$ rectangle r_i,we have the following 2-literal axiom clauses:

$$\neg px_{i,e} \vee px_{i,e+1}$$

$$\neg py_{i,f} \vee py_{i,f+1}$$

2.3.4 Non-overlapping Constraints 1

$\forall$ rectangle r_i, r_j (i < j), we have the following 4-literal clauses:

$$lr_{i,j} \vee lr_{j,i} \vee ud_{i,j} \vee ud_{j,i}$$

2.3.5 Non-overlapping Constraints 2

$\forall$ rectangle r_i, r_j (i < j), we also have the following 3-literal clauses:

$$\neg lr_{i,j} \vee px_{i,e} \vee \neg px_{j,e+w_i}$$

$$\neg lr_{j,i} \vee px_{j,e} \vee \neg px_{j,e+w_j}$$

$$\neg ud_{i,j} \vee py_{i,f} \vee \neg py_{j,f+h_i}$$

$$\neg ud_{j,i} \vee py_{j,f} \vee \neg py_{i,f+h_j}$$

3 2^{nd}model -create_model_with_rotation (Point 5)

The same previous model with handling the possible rotation of each presents/-pieces.

3.1 New Variables

rot: Array of 0|1 with dimension equal to the number of presents/pieces. If the i^{th} element of the array is *'0'* means no rotation otherwise if *'1'* that piece is rotated by 90°

$$\textbf{rot} \quad = [rot\_1, rot\_3, \ ... \ , rot\_n]$$

3.2 New Function

def **get_dim**(i, rot, d) : According to the rotation(rot), return the correct value of the width(*d=0*)/height(*d=1*) of a given piece index(i). This function is used every time I need to know the width/height of a piece, the pieces's dimension is not constant anymore like in the previous model.

3.3 New Constraints

In all constraints I no longer use the initial dimensions of the pieces but I use the new function *get_dim*.

3.3.1 square = no rotation

It is unnecessary to rotate any square piece (width = height) because I will get the same solution (same coordinates for each piece).

3.3.2 force at least one rotation

Not necessary but this constraint has been added to force on having a solution with at least one piece rotated, just for demonstration and visualization purpose.

4 3^{rd}model - create_model_same_dim (Point 6)

The same model as *create_model* but taking into consideration the fact that there can be multiple pieces with the same dimension. Also the previous models are able to solve instances in which there are several pieces with the same dimensions but in those cases it is possible to fix the positional relation with each other reducing the domains for some pieces and speeding up the instance resolution itself[1].

4.1 New Constraints

4.1.1 same dimension

If we have rectangles/pieces r_i, r_j and r_k which have the same dimensions we can fix/constrain the positional relation of those rectangles, between r_i-r_j and r_j-r_k. For example, if we have 2 pieces with the same dimension(2×2), the second piece(the pink one) will never placed to the left or under the first one(the yellow one). it will be placed always on the right or upper the first one.

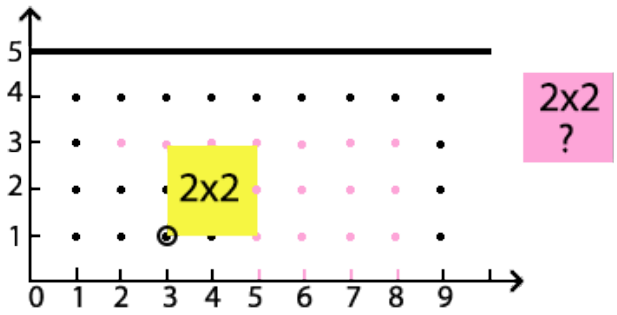

Figure 1: Positional relation constraint

Formally:

$\forall$ rectangle r_i, r_j (i < j, r_i_w=r_j_w, r_i_h=r_j_h), we can assign:

$$lr_{j,i} = false$$

$$lr_{i,j} \vee \neg ud_{j,i}$$

5 References

[1] Takehide, S., Katsumi, I., Naoyuki, T., Mutsunori, B., Hidetomo, N. *A SAT-based Method for Solving the Two-dimensional Strip Packing Problem.* http://ceur-ws.org/Vol-451/paper16soh.pdf

[2] Lesh, N., Marks. J., McMahon A., Mitzenmacher M. (2004) *Exhaustive approaches to 2D rectangular perfect packings.* https://www.eecs.harvard.edu/ michaelm/postscripts/ipl2004.pdf

[3] Simonis, H. & O'Sullivan, B. (2008) *Search Strategies for Rectangle Packing.* International Conference on Principles and Practice of Constraint Programming: pp 52-66.

[4] Clautiaux, F., Jouglet, A., Carlier, J., Moukrim, A. (2006) *A new constraint programming approach for the orthogonal packing problem.* http://vmk.ugatu.ac.ru/c%26p/article/new_2009/2D_OPP_clautiaux_constraint_progr.pdf

[5] Huang, E., Korf, R.E. (2012) *Optimal Rectangle Packing: An Absolute Placement Approach.* https://arxiv.org/ftp/arxiv/papers/1402/1402.0557.pdf