Course Project
Present Wrapping Problem

Daniele Domenichelli - 954277
daniele.domenichell2@studio.unibo.it

Nicola Amoriello - MATRICOLA
EMAIL

January 2021

Contents

Chapter 1

Introduction

The Present Wrapping Problem is...

Chapter 2

CP

A common scientific pattern, usually used to better understand a problem, is to decompose the case into simpler and simpler parts that take in account just one or few aspects of the problem. When we can control those aspect with a certain amount of reliability, we can mix different parts in order to ensure that the superposition of those effects behaves as expected. In this way we can build incremental models, that solve the problem by looking and optimizing a certain aspect if the problem.

2.1 Base Model

The **Base Model** is the most basic, where we defined our problem view, such as the parameters and the variables, and we decided how to constraint it in order to get a satisfiable solution:

Parameters		
Parameter	**Description**	
Width	The Paper Sheet Width	
Height	The Paper Sheet Height	
Presents	The number of the Presents to place in the Paper Sheet	
Dimension X	The array of the x dimensions of the Presents	
Dimension Y	The array of the y dimensions of the Presents	
Extracted Parameters		
Parameter	**Formula**	**Description**
Area	$Area = Width \cdot Height$	Area of the Paper
Areas	$Areas[i] = Dimension_x[i] \cdot Dimension_y[i]$	The array of the areas of the Presents
Variables		
Variable	**Description**	
Coord X	Array of the X positions of each Present	
Coord Y	Array of the Y positions of each Present	

2.1.1 Main Problem Constraints

Once the description of the problem is carried out, we defined some general constraints in order to instruct the way to find a solution to the solver. The constraints are:

Essential Constraints

- ***The presents must fit into the Paper Sheet:***

  A present fits in the paper if its coordinates are strictly positive and its coorinates summed with its corresponding dimensions are lesser then the Paper Sheet dimensions.

  The resultant constraint is:

  $\forall i \in [1, Presents] \rightarrow$
  $(Coord_x[i] + Dimension_x[i] \leq Width + 1) \wedge$
  $(Coord_y[i] + Dimension_y[i] \leq Height + 1)$

  As we used indexes starting from 1, we must add 1 to the right side of both disequations

- ***Two different presents must not overlap:***

  Given the two rectangles of two different presents, we can check if they have at least one part in common, just by checking their corners. So, we defined the *overlaps* predicate:

  $overlaps(Left_x^1, Right_x^1, Left_y^1, Right_y^1, Left_x^2, Right_x^2, Left_y^2, Right_y^2) \leftrightarrow$
  $\neg(Left_x^1 \geq Right_x^2 \vee Left_x^2 \geq Right_x^1) \wedge$
  $\neg(Right_y^1 \leq Left_y^2 \vee Right_y^2 \leq Left_y^1)$

  Each present is described as the rectangle:

  $Left_x^i, Left_y^i, Right_x^i, Right_y^i$

  So we can constraint each couple of presents to not overlaps one to each other:

  $\forall i, j \in [1, Presents], j > i \rightarrow$
  $\neg overlaps($
  $Coord_x[i], Coord_x[i]{+}Dimension_x[i], Coord_y[i], Coord_y[i]{+}Dimension_y[i],$
  $Coord_x[j], Coord_x[j]{+}Dimension_x[j], Coord_y[j], Coord_y[j]{+}Dimension_y[j]$
  $)$

Additional Constraints

These constraint are not essential to solve the general formulation of this problem, but they results helpful as they restrict the search space in the given instances. The underlying assumption is that the instance contains the right amount of presents such that the area of the Paper Sheet is completely used.

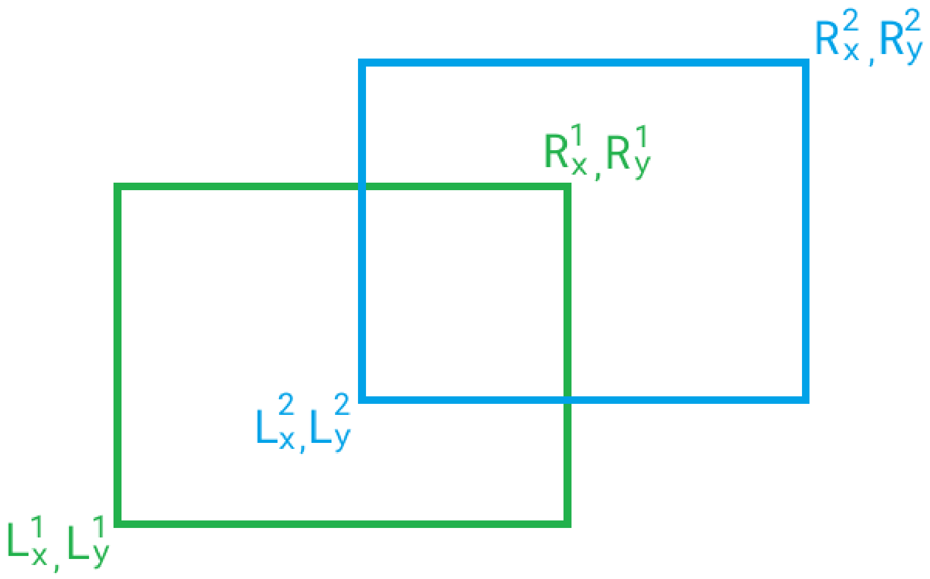

Figure 2.1: Overlapping Model

- ***The total area of the presents must be the same of the Paper Sheet:***

  $\sum_{i=1}^{Presents} Areas[i] = Area$

  This constraint prevents the exploration of the search space at the very beginning. We indeed can instantly infer if the given instance is feasible: if the total areas does not match we can say the problem is unsatisfiable.

  A further relaxation of this constraint is to use $\leq$ instead of $=$ in order to keep instances where we have presents that do not completely fill the Paper Sheet. We kept the strict constraint for efficency reason, because the given instances all fall in this case.

- ***The presents must fill the row (column) dimension:***

  As an extension of the previous constraint, we want to use each row *(or column)* such that we use all of the available area of the paper.

  Drawing a vertical *(horizontal)* line and summing up the encountered presents dimensions we must end up with the same dimension of the Paper Sheet:

  Rows: $\forall y \in [1, Height] \rightarrow$

  $$\sum_{i=1}^{Presents} \begin{cases} Dimension_x[i] & \text{if } y \geq Coord_y[i] \land y < Coord_y[i] + Dimension_y[i] \\ 0 & \text{otherwise} \end{cases}$$

  $= Width$

Columns: $\forall x \in [1, Width] \rightarrow$
$$\sum_{i=1}^{Presents} \begin{cases} Dimension_y[i] & \text{if } x \geq Coord_x[i] \wedge x < Coord_x[i] + Dimension_x[i] \\ 0 & \text{otherwise} \end{cases}$$
$= Height$

2.1.2 Search Methods

All of the constraints we described so far could solve the given instances with the *Geocode* solver, but the main difficulty is the time spent in the resolution. Some instances can take more then 10 minutes. To lower the elasped time, we can tell to the solver how to optimize the search on the variables:

- We decided to choose a preferential axes for the search. The X axis was choosen.
- Each axis then can be explored in different ways. We want to explore it with the most difficult case as we already know that some presents configurations can exclude a priori the placement of other presents. In this way we selected the ***first_fail*** search parameter, that chooses the variable with the smallest domain and try to find out if can have a value in the current solution state. If there are no possible values, we prevented the solver to search useless branch of the search tree. As we place presents into the sheet, each variable will lose a part of its domain, so we will choose that one that is most likely to fail.
- Now we must select an heuristic that chooses intelligently a value for the given variable. Our problem description has coordinates of each presents in their lower left corner, so we try to assign first the lesser available coordinates, then the bigger one. The ***indomain_min*** search parameter try to assign to each variable the minimum value available in the current domain.
- The final search annotation is:

  $seq\_search([$
  $\quad int\_search(Coord\_X, first\_fail, indomain\_min),$
  $\quad int\_search(Coord\_Y, first\_fail, indomain\_min)$
  $])$

We also tried any combination of all the possible parameters in order to confirm our reasoning, so we end up by choosing this setup because it resulted the most performant.

2.2 Symmmetry Model

We had further analysed the problem in order to understand if, from an erroneous solution, there are similar solutions that we can deduce as unsatisfiable

as they are permutation or simmetrical of the erroneous one. This technique is called **Symmetry Breaking**.
The **Present Wrapping Problem** [1] is an extension of the **2D Bin Packing Problem**, and one of the most effective heuristic to place presents is to choose those that are more restricting for the others, in other words, the bigger the present is, the most difficult is to place, the more it will restrict the other presents domains and the more effective will be its placement in the first stages. So the best analytical and empirical heuristic found so far for this kind of problem is to sort the presents in size order, placing the bigger first and the smaller last [2, 3].
Doing this requires a new extracted parameter:

Extracted Parameters		
Parameter	**Formula**	**Description**
Sorted Areas Indexes	$Sorted\_Areas\_Indexes = reverse(arg\_sort(Areas))$	Indexes of the Areas sorted by Present Area

This new parameter stores the indexes of the sorted areas, so the $Sorted\_Areas\_Indexes[1]$ will store the indexes of the present with the maximum area, $Sorted\_Areas\_Indexes[2]$ the index of the second present with maximum area and so on.
Now the most basic constraint we can add is that the biggest present will always stay on the minimal coordinates:
$Coord\_X[Sorted\_Areas\_Indexes[1]] = 1$
$Coord\_Y[Sorted\_Areas\_Indexes[1]] = 1$

Then we want to place the bigger presents in the left-bottom most part of the paper, simulating the fact that we are placing them before the others:
$\forall i, j \in [1, Presents], j > i \rightarrow$
$Coord_y[Sorted\_Areas_Indexes[i]] = Coord_y[Sorted\_Areas\_Indexes[j]] \rightarrow$
$Coord_x[Sorted\_Areas_Indexes[i]] < Coord_x[Sorted\_Areas\_Indexes[j]]$
This, in combination with the search method, provides that the bigger present will be then the lesser will be its coordinate x, and since the bigger the present, the smaller is its domain, it will be also placed first, that means in the lower y possible. By doing this we can exclude all the possible symmetries due to the swap of different area presents.
Excluding the symmetrical solutions allow us to exclude also the symmetrical part of the search tree that are unsatisfiable, just by finding an unsatisfiable combination out of the all simmetricals.

2.3 Rotation Model

2.4 Symmetry Rotation Model

As we growth the model in modules, we can just combine the **Symmetry Model** with the **Rotation Model** and we end up with a **Symmetry Rotation**

Model that takes in account the possibility of the presents rotation and also excludes the symmetrical solutions.

2.5 Duplicated Symmetry Model

Another point to take in account, is the possibility of the presence of presents that have the same size. As we modelled the problem, the **Base Model** can already solve this kind of instances, but we can add some constraints in order to exploit the Symmetry Breaking even in these cases.

2.6 Duplicated Symmetry Rotation Model

The modularity of our model eassily achieves a new model that takes in account all the discussed properties of the problem *(Symmetry, Rotation, Duplicated Presents)* at once, just by combining the constraints of all the precedent models. The results show that this model achieve the best performance, as the number of errors and the quantity of the explorated nodes in the search tree drastically decrease.

2.7 Remarks

Chapter 3

SMT

In this chapter we are going to cover the solution of the Present Wrapping Problem using Satisfiability Modulo Theories (SMT) with the help of tools such as Z3 python API [4] and SMT2LIB [5] standard language.
To better explore the problem and all the possible solutions we decided to create a model for each approach so as to be able to understand the effects more easily and only at the end incorporate everything that was learned in the intermediate stages.

3.1 Base Model

The baseline model is the simplest model we have implemented and also includes all the parameters, variables and constraints on which all subsequent models are based.
The parameters and variables used are the same as those already defined:

Parameters		
Parameter	**Description**	
Width	The Paper Sheet Width	
Height	The Paper Sheet Height	
Presents	The number of the Presents to place in the Paper Sheet	
Dimension X	The array of the x dimensions of the Presents	
Dimension Y	The array of the y dimensions of the Presents	
Extracted Parameters		
Parameter	**Formula**	**Description**
Area	$Area = Width \cdot Height$	Area of the Paper
Areas	$Areas[i] = Dimension_x[i] \cdot Dimension_y[i]$	The array of the areas of the Presents
Variables		
Variable	**Description**	
Coord X	Array of the X positions of each Present	
Coord Y	Array of the Y positions of each Present	

3.1.1 Main Problem Constraints

We need to define constraints that are able to give valid instructions to the solver so that we can return a valid solution to the problem we are facing.

Essential Constraints
First of all we need to define the constraints that allow us to have only valid solutions as output: that is, all those constraints that define the problem treated together with the parameters and variables previously discussed.
The following is a list of these required constraints:

- ***The presents must fit into the Paper Sheet:***

  Obviously a present has a certain size (both in width and in height) which must be a positive number and which must not exceed the size of the paper in which it is to be placed.

  The resultant constraint is:

  $\forall i \in [1, Presents] \rightarrow$
  $(Coord_x[i] + Dimension_x[i] \leq Width + 1) \wedge$
  $(Coord_y[i] + Dimension_y[i] \leq Height + 1)$

  As we used indexes starting from 1, we must add 1 to the right side of both disequations

- ***Two different presents must not overlap:***

  The other essential constraint is about the not overlap principle.
  Through the `overlaps` function defined by us we can pass as parameters the indices of the two distinct presents of which we want to know if they overlap each other or not.

  Knowing the two rectangles taken into consideration we can easily understand if these two overlap at least in one point by comparing the spatial coordinates of the horizontal and vertical boundaries of the two.
  Here how we defined the *overlaps* constraint in a mathematical way:

  $overlaps(Left^1_x, Right^1_x, Left^1_y, Right^1_y, Left^2_x, Right^2_x, Left^2_y, Right^2_y) \leftrightarrow$
  $\neg(Left^1_x \geq Right^2_x \vee Left^2_x \geq Right^1_x) \wedge$
  $\neg(Right^1_y \leq Left^2_y \vee Right^2_y \leq Left^1_y)$

  Where $Left^i_x, Left^i_y, Right^i_x, Right^i_y$ are the present spacial coordinate.

  By means of this we can check in pairs if the ragals do not overlap each other:

  $\forall i, j \in [1, Presents], j > i \rightarrow$
  $\neg overlaps($
  $Coord_x[i], Coord_x[i]{+}Dimension_x[i], Coord_y[i], Coord_y[i]{+}Dimension_y[i],$

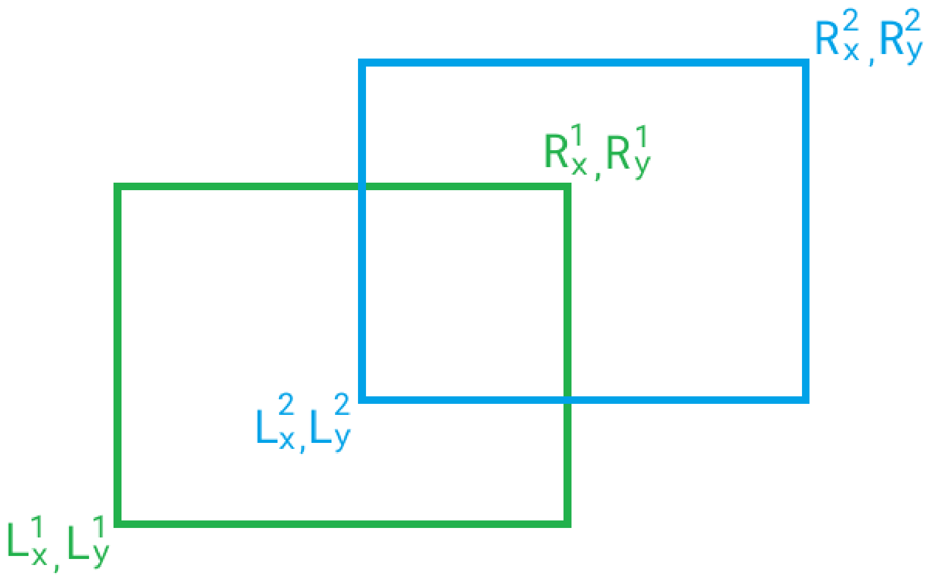

Figure 3.1: Overlapping Model

$Coord_x[j], Coord_x[j]+Dimension_x[j], Coord_y[j], Coord_y[j]+Dimension_y[j]$
)

Additional Constraints

In addition to the previous constraints, which are inevitable for the correct definition of the problem, we have decided to implement further constraints to restrict the domain of possible solutions and make the solver more efficient.

- ***The total area of the presents must be the same of the Paper Sheet:***

  $\sum_{i=1}^{Presents} Areas[i] = Area$

  Thanks to this constraint we can understand from the beginning of the search if the given instance is feasible or not: in this way we can avoid the search a priori and avoid a waste of resources in case of unfeasibility.

  A further relaxation of this constraint is to use $\leq$ instead of $=$ in order to keep instances where we have presents that do not completely fill the Paper Sheet. We kept the strict constraint for efficency reason, because the given instances all fall in this case.

- ***The presents must fill the row (column) dimension:***

A further step to optimize our solver was to add a constraint where it is checked whether each row *(column)* is filled completely along its width *(height)*.

Here follow the two different definitions of this constraint:

Rows: $\forall y \in [1, Height] \rightarrow$
$\sum_{i=1}^{Presents} \begin{cases} Dimension_x[i] & \text{if } y \geq Coord_y[i] \wedge y < Coord_y[i] + Dimension_y[i] \\ 0 & \text{otherwise} \end{cases}$
$= Width$

Columns: $\forall x \in [1, Width] \rightarrow$
$\sum_{i=1}^{Presents} \begin{cases} Dimension_y[i] & \text{if } x \geq Coord_x[i] \wedge x < Coord_x[i] + Dimension_x[i] \\ 0 & \text{otherwise} \end{cases}$
$= Height$

3.2 Symmmetry Model

As has already been done for the implementation in CP, also here we have decided to apply a similar method of **symmetry breaking** to remove rotated or mirrored solutions. To do this we used the heuristic to select the most voluminous presents (in this case we intend those with the largest area) first and place them in the lowest-left available place [2, 3]. This allows us to always work in the lower left quadrant so as to avoid specular solutions that differ only from the reference quadrant.
As in the analogous model for CP, here too we have extracted the *"Sorted Area Indexes"* parameter, which is essential to implement the heuristics just described:

Extracted Parameters		
Parameter	**Formula**	**Description**
Sorted Areas Indexes	$Sorted\_Areas\_Indexes = reverse(arg\_sort(Areas))$	Indexes of the Areas sorted by Present Area

In the *"Sorted Area Indexes"* parameter, as can be seen from the name, is a list with the indices of the gifts arranged in ascending order with respect to the area. In this way we can easily define that the first object of the list should be placed first in the lower-left corner of our paper in a hard-coded way:

$Coord\_X[Sorted\_Areas\_Indexes[1]] = 1$
$Coord\_Y[Sorted\_Areas\_Indexes[1]] = 1$

In the same way we can go have all of the following presents in the list in order to respect the rule "the biggest first":

$\forall i, j \in [1, Presents], j > i \rightarrow$
$Coord_y[Sorted\_Areas_Indexes[i]] = Coord_y[Sorted\_Areas\_Indexes[j]] \rightarrow$

$Coord_x[Sorted\_Areas_Indexes[i]] < Coord_x[Sorted\_Areas\_Indexes[j]]$

3.3 Rotation Model

In order to expand our model so that it is possible to rotate a block, thus having further solutions to explore in our problem, we needed to add a new *"rotated"* variable:

Variables	
Variable	**Description**
Rotated	The boolean that indicates whether a present is rotated or not

If *"rotated"* were set to True, the dimensions X and Y would be swapped to represent the present rotated by 90 ° (or 270 °). In the False case, the dimensions remain unchanged and represent the object not rotated (or rotated by 180 °). All this is easily implemented with a boolean check when returning the dimensions of a single present. Here follows the definition of what just described:

$$Get\_Dimension_x = \begin{cases} Dimension_y & \text{if } Rotated \\ Dimension_x & \text{otherwise} \end{cases}$$

$$Get\_Dimension_y = \begin{cases} Dimension_x & \text{if } Rotated \\ Dimension_y & \text{otherwise} \end{cases}$$

3.4 Symmetry Rotation Model

Following as done in CP, also in SMT we decided to combine the characteristics of the previously implemented models.
We merge together the **Symmetry Model** with the **Rotation Model** and we made the **Symmetry Rotation Model** that takes in account the possibility of the presents rotation and also excludes the symmetrical solutions.

3.5 Duplicated Symmetry Model

Another point to take in account, is the possibility of the presence of presents that have the same size. As we modelled the problem, the **Base Model** can already solve this kind of instances, but we can add some constraints in order to exploit the Symmetry Breaking even in these cases.

3.6 Duplicated Symmetry Rotation Model

This model simply incorporates all the features implemented in the previous models *(Symmetry, Rotation, Duplicated Presents)*.
In this way it is possible to benefit at the same time from features, such as

rotation and the distinction between two different gifts of the same size, and from symmetry breaking to remove rotated and mirrored solutions from the domain.

3.7 Remarks

Here are the results obtained from the various SMT models described in this chapter:

Chapter 4

SAT

4.1 Base Model

4.1.1 Main Problem Constraints

4.2 Rotation Model

4.3 Remarks

Chapter 5

Conclusions and Remarks

Bibliography

[1] Combinatorial Decision Making and Optimization - Project Description

[2] Code inComplete - Binary Tree Bin Packing Algorithm https://codeincomplete.com/articles/bin-packing/

[3] The Algorithm Design Manual - Steven S S. Skiena

[4] Z3Prover/z3 - The Z3 Theorem Prover https://github.com/Z3Prover/z3

[5] SMT-LIB http://smtlib.cs.uiowa.edu/