Course Project
Present Wrapping Problem

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

Nicola Amoriello - 952269
nicola.amoriello@studio.unibo.it

January 2021

Contents

Chapter 1

Introduction

*"It [The Present Wrapping Problem] is a common practice that a private business rewards its loyal clients with presents, which are typically wrapped in a costly corporate paper covered with the logo of the business. Imagine that you work for such a business which wants to limit the overall amount of paper that can be used for this purpose, in order to reduce the associated expenses." [**?**]*

In the following report we describe our solutions to the proposed problem. We develop many strategies exploting different techniques such as **Constraint Progamming *CP***, **Satisfiability Modulo Theories *(SMT)*** and also **Boolean Satisfiability *(SAT)***.
The problem is a derivation of the general problem called as **Bin Packing Problem** [**?**], where a certain ammount of blocks must fit in a bounded weighted space, without overlapping each other. In this particular case, our blocks are represented by presents belonging a 2D discrete space, represented by the gift paper sheet. Our target is to check if a given ammount of presents, with certain dimensions, can fit into a fixed size paper sheet.

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.

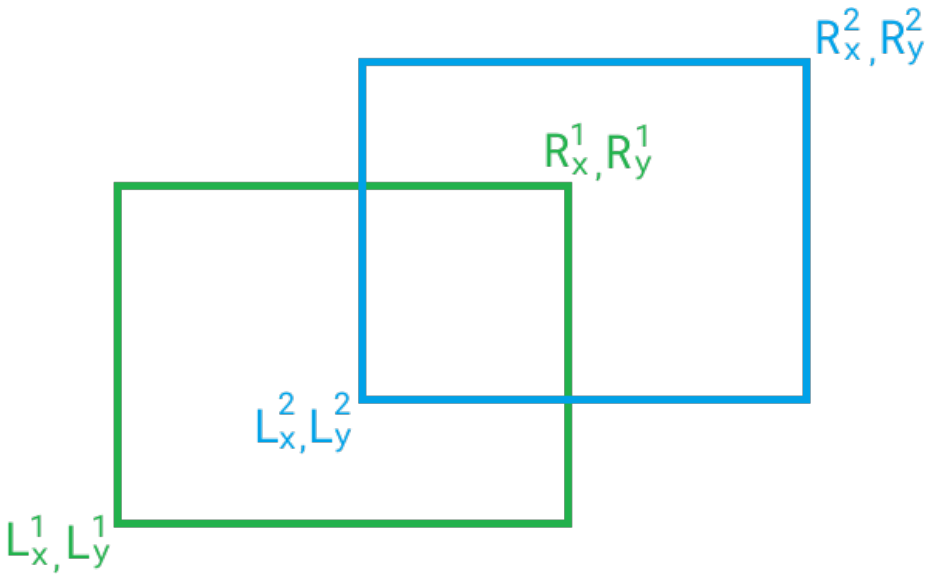

Figure 2.1: Overlapping Model

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

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

  $= Width$

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

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 annotiation 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.

2.1.3 Results

Results			
Instance	**Time *[s]***	**Nodes**	**Propagations**

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** [**?**] 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 [**?**, **?**].
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.

Results			
Instance	**Time *[s]***	**Nodes**	**Propagations**

2.3 Rotation Model

In a real life case we just know the two dimensions of each present we want to place, but we dont know in which order they should appear such that we can fit the paper sheet. The rotation model can overwhelm this problem because it looks for any combination of rotated presents over the paper sheet, so we don't need to specify the right combination of dimensions that can fit the paper. In order to do this, we need another variable in our description:

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

This variable keep trace of the rotation of the present. Keep in mind that in a discretized space, we can rotate a rectangular present just in two direction: 0 deg or 90 deg. Indeed if we further rotate the present, 180 deg for example, we end up with the non rotated present, or even more at 270 deg we obtain the 90 deg rotated present. Thanks to their regularity of the geometric shape of the presents there are only two conditions of rotation, described by the inversion of the two dimensions. To keep the problem description as simple as possible, we can just create a proxy function that returns the correct dimension depending on its rotation. So if the present is not rotated, it return the right dimension, otherwise it will return the opposite dimension:

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

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

Now, we can change any constraint that involves a dimension variable with the corresponding proxy. In this way we obtained a model that can solve instances of the problem that are satisfiable only if we rotate one *(or more)* present.

Results			
Instance	**Time *[s]***	**Nodes**	**Propagations**
8x8	0.001	9	658
9x9	0.002	10	1210
10x10	0.002	12	1528
11x11	0.004	36	6827
12x12	0.003	27	4698
13x13	0.003	31	5859
14x14	0.016	43	8206
15x15	0.008	37	11236
16x16	0.004	23	6360
17x17	0.007	34	10908
18x18	4.728	12027	6548570
19x19	0.126	304	107100
20x20	0.073	297	116408
21x21	0.073	257	90569
22x22	0.027	68	55549
23x23	16.056	24043	18445778
24x24	0.778	1483	1109728
25x25	0.336	819	591748
26x26	1.123	2534	1470455
27x27	5.310	6470	8463102
28x28	3.132	4165	4817887
29x29	3.082	3683	4938340
30x30	0.275	611	514051
31x31	1.453	2362	2419560
32x32	14.042	13639	25412055
33x33	0.320	531	373851
34x34	5.510	6336	9279861
35x35	2.962	4036	5119177
36x36	27.857	23856	44708965
37x37	4.729	7786	5724185
38x38	0.447	848	1045461
39x39	1062.707	613863	1235535096
40x40	10.516	10687	14672008
rotation$_t est$	0.000	8	422

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.

Results			
Instance	**Time *[s]***	**Nodes**	**Propagations**

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. The simpliest approach is to force the same size presents to be placed in the order they appear. In this way we put in the lesser coordinates the presents that are in the first positions of the parameter $Dimension_X$ and $Dimension_y$ arrays:

$$\forall i, j \in [1, Presents], j > i \rightarrow$$
$$Dimension_x[Sorted\_Areas\_Indexes[i]] \neq Dimension_x[Sorted\_Areas\_Indexes[j]] \wedge$$
$$Dimension_y[Sorted\_Areas\_Indexes[i]] \neq Dimension_y[Sorted\_Areas\_Indexes[j]] \wedge$$
$$Coord_y[Sorted\_Areas\_Indexes[i]] \leq Coord_y[Sorted\_Areas\_Indexes[j]]$$

In this formula we are exploiting the search method, indeed we do not need to constrain the X coordinates because the **first_fail** approach do it for us. Furthermore, we decided to use the already sorted areas array for efficiency reasons, because the same size presents will appear in near positions in that array, while they could appear in distant positions in the non-sorted one.

Results			
Instance	**Time *[s]***	**Nodes**	**Propagations**

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.

Results			
Instance	**Time *[s]***	**Nodes**	**Propagations**

2.7 Global Constraints Model

For the study case, we choose to try to implement our constraints through the already defined MiniZinc global constraints:

- The `overlaps` predicate can well be substituted by the `diffn` global constraint. Furthermore, the latter can work directly on arrays so the new constraint will be just one line of code:

  $diffn(Coord_x, Coord_y, Dimension_x, Dimension_y)$

- The *fit row/col* constriants can be substituted by the `cumulative` global constraint:

  Rows: $cumulative(Coord_x, Dimension_x, Dimension_y, Height)$

  Cols: $cumulative(Coord_y, Dimension_y, Dimension_x, Width)$

  Unluckly this global constraint was thought for task scheduling problems, so the performance result are not so good at all.

- The Duplicated **Symmetry Breaking** constraint can also be replaced by the `lexlesseq` global constraint:

  $lexlesseq(Sorted\_Areas\_Coord_y, Coord_y)$

  With $Sorted\_Areas\_Coord_y$ is the array of $Coord_y$ accessed with the indexes of the $Sorted\_Areas\_Indexes$ array.

At the end, we choosed to stuck with our implementation because it was well optimized for this kind of problem, and results to be more efficient in terms of time, during the resolution of big size problems.

Results			
Instance	**Time *[s]***	**Nodes**	**Propagations**

2.8 Remarks and Results

As MiniZinc is an high level interface for many solver, we tryied different solver configurations in order to understand which one performs better in our problem. The standard `Geocode` solver resulted well suitable for any given instance, but we found out that the best solver, in particular for the bigger instances, was the `Chuffed` solver. The latter indeede exploit some **SAT** techniques to better explore and learn wrong or symmetric pattern in the search space in order to prevent the exploration of useles nodes and branches.

We briefly recap the overall results of the previous models in a textual informative table:

Global Results				
Model	**Speed**	**Complexity**	**Strengths**	**Weaknesses**

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 [**?**] and SMT2LIB [**?**] 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:

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

  Where $Left_x^i, Left_y^i, Right_x^i, Right_y^i$ are the present spacial coordinate.

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

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

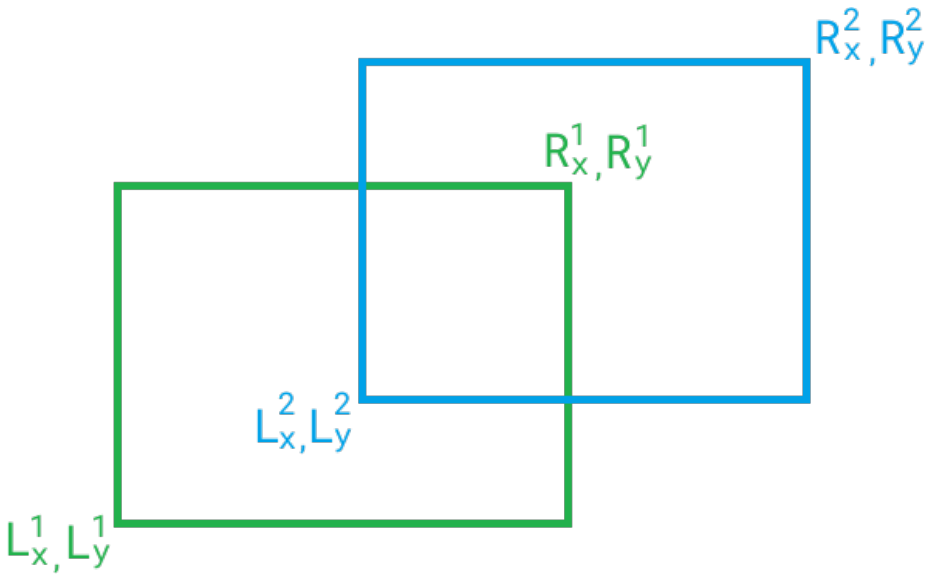

Figure 3.1: Overlapping Model

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

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: $\forall x \in [1, Width] \rightarrow$

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

$$= Height$$

3.1.2 Results

Results			
Instance	**Time *[s]***	**Nodes**	**Propagations**
8x8	0.091	8	221
9x9	0.079	25	307
10x10	0.079	44	954
11x11	0.157	148	5707
12x12	0.162	214	5214
13x13	0.084	54	1119
14x14	0.160	84	5006
15x15	0.311	534	14617
16x16	0.265	520	13157
17x17	0.360	1056	51701
18x18	0.667	1963	143574
19x19	0.471	1981	98414
20x20	1.140	3596	336263
21x21	1.380	4962	350477
22x22	1.234	4391	273473
23x23	13.028	9140	496203
24x24	2.947	7538	744512
25x25	2.635	8611	571592
26x26	36.027	24456	2294296
27x27	17.076	11429	1054566
28x28	49.265	33951	5081383
29x29	65.091	49908	7666862
30x30	29.837	11575	1098465
31x31	3.479	8155	613409
32x32	119.407	80793	16891820
33x33	63.708	50550	7659959
34x34	21.128	15144	1456046
35x35	43.466	31302	5072455
36x36	45.478	39013	5193048
37x37	84.938	115989	25947324
38x38	5.099	7602	865820
39x39	214.940	136091	31038317
40x40	22.037	15290	2315027
rotation$_t est$	-	-	-

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 [**?**, **?**]. 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":

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

Results			
Instance	**Time *[s]***	**Nodes**	**Propagations**

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 array 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:

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

Results			
Instance	**Time *[s]***	**Nodes**	**Propagations**

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.

Results			
Instance	**Time *[s]***	**Nodes**	**Propagations**

3.5 Duplicated Symmetry Model

As we did in the CP models, we can model those instances that have presents with the same dimensions. As we modelled the problem, the **Base Model** can already solve this kind of instances, but we can add some constraints to take in account symmterical solutions. The simpliest approach is to force the same size presents to be placed in the order they appear. In this way we put in the lesser coordinates the presents that are in the first positions of the parameter $Dimension_X$ and $Dimension_y$ arrays:

$$\forall i, j \in [1, Presents], j > i \rightarrow$$
$$Dimension_x[Sorted\_Areas\_Indexes[i]] \neq Dimension_x[Sorted\_Areas\_Indexes[j]] \wedge$$
$$Dimension_y[Sorted\_Areas\_Indexes[i]] \neq Dimension_y[Sorted\_Areas\_Indexes[j]] \wedge$$
$$Coord_y[Sorted\_Areas\_Indexes[i]] \leq Coord_y[Sorted\_Areas\_Indexes[j]]$$

By adding this constraint, we force the solver to exclude the solutions where the same size presents can swap each other, just by forcing the solver to put them in the lesser coordinates possible as before they appear in the parameter dimensions array.

Results			
Instance	**Time *[s]***	**Nodes**	**Propagations**

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.

Results			
Instance	**Time** *[s]*	**Nodes**	**Propagations**

3.7 Remarks and Results

We briefly recap the overall results of the previous models in a textual informative table:

Global Results				
Model	**Speed**	**Complexity**	**Strengths**	**Weaknesses**

Chapter 4

SAT

The **Boolean Satisfiability** can be exploited in order to prove that the given ammount of presents, with the given dimensions can fit in certain positions into the paper sheet. As far we have not numerical variables anymore we must reimplement from scratch the whole models definition. We borrowed some concepts from the **CP** and **SMT** methods, but we had to port them into a new boolean logic.

4.1 Base Model

This model is the porting of the **SMT Base Model**, but we must describe the coordinates system with another variable. Indeed, we loose all the variables of the precedent model, and we use a new tensor that will describe the whole problem.

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**	
Paper	A 3D boolean tensor describing the presence of the present in a particular position	

The *Paper* tensor has two dimensions for indicating the present position and one dimension indicating the present index. In this way we know that the i-th

present will occupy the cell in the coordinates x, y if the boolean value of the $tensor[x, y, i]$ is true.

4.1.1 Main Problem Constraints

Now that the problem variables are decided, we can constraint the *Paper* with some predicates, in **Propositional Logic**, in order to carry out the solution of the problem.

Essential Constraints

- ***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 if the tensor at position (x, y) holds in two different presents i and j. The *overlaps* predicate is defined as:

$$overlaps(Present_1, Present_2) \leftrightarrow \bigvee_{x,y \in Paper} (Paper[x, y, Present_1] \wedge Paper[x, y, Present_2])$$

- ***The presents must have and occupy the correct dimension:***

  This was one of the hardes constrain to develop. We have to force the tensor to have the right ammount of true values in the correct place, for each present at a gien coordinate. The idea is that given a certain coordinate, we force the tensor to obbey a certain *Disjunctive Normal Formula*.

  For each present, we fix a tuple of initial coordinates (x_0, y_0) and we force the tensor to hold at all the subsequent $Width \times Height$ coordinates, and not to hold the rest. Then we translate the initial coordinates and repeat the extraction of the formula. Once we have all the formulas for all the possible initial position of the present in the paper sheet, we concatenate them with an Or series into a *Disjunctive Normal Formula*:

$$\bigwedge 2_{p \in [1, Presents]} \bigvee_{\substack{x_0 \in [1, Width - Dimension_x[p]] \\ y_0 \in [1, Height - Dimension_y[p]]}} (\bigwedge 2_{\substack{x \in [x_0, Dimension_x[p]] \\ y \in [y_0, Dimension_y[p]]}} Paper[x, y, p]) \vee (\bigwedge 2_{\substack{x \in [1, x_0] \cup [x_0 + Dimension_x[p], Width] \\ y \in [1, y_0] \cup [y_0 + Dimension_y[p], Height]}} \neg Paper[x, y, p])$$

- ***Each tensor tuple of coordinates must have at least one present:***

We want the tensor to have at least one present at each tuple of coordinates (x, y):

$$\bigwedge 2_{\substack{x\in[1,Width]\\y\in[1,Height]}} \bigvee_{p\in[1,Presents]} Paper[x,y,p]$$

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.

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

  We want to use each row *(or column)* such that we use all of the available area of the paper.

  Drawing a vertical *(horizontal)* we check that at least one present holds in the tensor in the line coordinates:

  Rows:

  $$\bigvee_{y\in[1,Height]} \bigwedge 2_{x\in[1,Width]} \bigvee_{p\in[1,Presents]} Paper[x,y,p]$$

  Cols:

  $$\bigvee_{x\in[1,Width]} \bigwedge 2_{y\in[1,Height]} \bigvee_{p\in[1,Presents]} Paper[x,y,p]$$

4.1.2 Results

Results			
Instance	**Time *[s]***	**Nodes**	**Propagations**

4.2 Rotation Model

Results			
Instance	**Time *[s]***	**Nodes**	**Propagations**

4.3 Remarks and Results

There are just a few of the implemented model because we wanted to devolop them just by using the Popositional Logic predicates, without recurring with Arithmetics and Numerical calculus.

We briefly recap the overall results of the previous models in a textual informative table:

Global Results				
Model	**Speed**	**Complexity**	**Strengths**	**Weaknesses**

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/`