Course Project
Present Wrapping Problem

Computer Science Department - Science and Engineering
Artificial Intelligence
University of Bologna

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

Daniele Domenichelli - 954277
daniele.domenichell2@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." [1]

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** [2], 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.

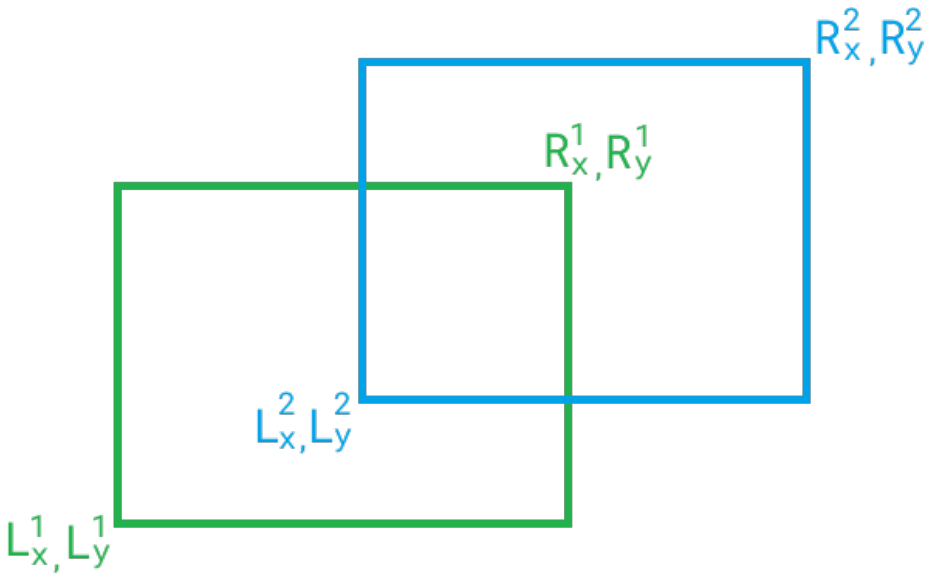

Figure 2.1: Overlapping Model

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

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

  $= Width$

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

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:

$$\begin{array}{l} seq\_search([\\ \quad int\_search(Coord\_X, first\_fail, indomain\_min), \\ \quad int\_search(Coord\_Y, first\_fail, indomain\_min) \\ ]) \end{array}$$

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.

Results			
Instance	**Time**	**Nodes**	**Propagations**
8x8	00:00:00.001	5	179
9x9	00:00:00.001	6	287
10x10	00:00:00.000	6	405
11x11	00:00:00.000	10	705
12x12	00:00:00.001	14	1,328
13x13	00:00:00.002	15	1,424
14x14	00:00:00.001	11	985
15x15	00:00:00.001	13	1,118
16x16	00:00:00.001	12	1,272
17x17	00:00:00.004	48	5,825
18x18	00:00:00.021	258	49,511
19x19	00:00:00.002	17	2,481
20x20	00:00:00.023	247	48,836
21x21	00:00:00.002	24	3,189
22x22	00:00:00.175	1,658	343,900
23x23	00:00:00.270	2,252	604,184
24x24	00:00:00.003	24	4,087
25x25	00:00:00.205	1,717	358,523
26x26	00:00:00.293	2,644	697,160
27x27	00:00:00.006	40	9,131
28x28	00:00:00.038	348	87,336
29x29	00:00:00.039	310	81,966
30x30	00:00:00.007	58	12,387
31x31	00:00:00.005	28	5,009
32x32	00:00:00.073	497	109,458
33x33	00:00:12.058	43,163	13,179,446
34x34	00:00:00.010	103	17,543
35x35	00:00:00.009	37	8,248
36x36	00:00:00.007	35	8,020
37x37	00:00:20.295	61,331	20,748,067
38x38	00:00:00.141	1,165	306,900
39x39	00:00:00.061	298	108,612
40x40	00:00:00.009	31	6,054
rotation_test	-	-	-

2.2 Symmetry 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 Pack-**

ing 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.

Results			
Instance	**Time**	**Nodes**	**Propagations**
8x8	00:00:00.001	3	150
9x9	00:00:00.001	5	357
10x10	00:00:00.003	5	461
11x11	00:00:00.001	10	1,265
12x12	00:00:00.007	45	8,116
13x13	00:00:00.004	11	2,159
14x14	00:00:00.001	19	2,599
15x15	00:00:00.011	142	27,081
16x16	00:00:00.002	9	1,814
17x17	00:00:00.011	80	24,551
18x18	00:00:00.003	26	5,218
19x19	00:00:00.039	316	91,564
20x20	00:00:00.099	536	123,172
21x21	00:00:00.192	900	304,328
22x22	00:00:00.125	544	190,132
23x23	00:00:06.933	19,260	9,171,901
24x24	00:00:00.371	1,488	638,669
25x25	00:00:03.943	12,490	4,788,188
26x26	00:00:01.367	3,595	2,087,114
27x27	00:00:02.178	4,784	3,049,562
28x28	00:00:16.155	43,295	23,048,023
29x29	00:00:05.314	15,069	11,628,851
30x30	00:00:00.046	287	100,741
31x31	00:00:00.135	916	301,542
32x32	00:00:00.148	387	267,579
33x33	00:00:00.350	1,580	770,610
34x34	00:00:00.471	1,701	827,950
35x35	00:00:00.410	1,752	935,117
36x36	00:00:04.800	14,412	8,047,884
37x37	00:00:39.646	93,562	45,084,879
38x38	00:00:01.475	6,150	2,711,326
39x39	00:00:04.727	16,581	10,166,321
40x40	00:00:02.233	7,408	3,362,652
rotation_test	-	-	-

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**	**Nodes**	**Propagations**
8x8	00:00:00.001	9	658
9x9	00:00:00.001	10	1,210
10x10	00:00:00.001	12	1,528
11x11	00:00:00.004	36	6,827
12x12	00:00:00.002	27	4,698
13x13	00:00:00.003	31	5,859
14x14	00:00:00.006	43	8,206
15x15	00:00:00.007	37	11,236
16x16	00:00:00.003	23	6,360
17x17	00:00:00.004	34	10,908
18x18	00:00:00.219	1,765	595,481
19x19	00:00:00.036	323	110,395
20x20	00:00:00.034	317	120,480
21x21	00:00:00.022	257	90,569
22x22	00:00:00.013	68	55,549
23x23	00:00:05.820	18,963	9,928,170
24x24	00:00:01.671	6,876	3,466,041
25x25	00:00:00.321	1,594	940,941
26x26	00:00:04.605	10,965	5,739,120
27x27	00:00:08.806	21,462	14,169,523
28x28	00:00:09.925	23,327	16,753,958
29x29	00:00:06.659	17,029	11,653,121
30x30	00:00:00.481	1,402	923,532
31x31	00:00:02.615	7,287	5,090,515
32x32	00:01:21.626	124,826	106,149,654
33x33	00:00:00.140	531	373,851
34x34	00:00:16.122	33,856	25,006,238
35x35	00:00:08.576	16,524	11,766,051
36x36	00:01:39.790	138,777	123,080,912
37x37	00:00:31.674	58,804	42,774,026
38x38	00:00:00.683	1,636	1,383,358
39x39	00:02:28.920	201,895	180,321,315
40x40	00:00:02.876	5,296	3,632,636
rotation_test	00:00:00.000	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**	**Nodes**	**Propagations**
8x8	00:00:00.001	6	617
9x9	00:00:00.001	8	1,209
10x10	00:00:00.004	9	1,614
11x11	00:00:00.002	15	3,303
12x12	00:00:00.020	98	41,544
13x13	00:00:00.006	29	9,686
14x14	00:00:00.007	25	8,707
15x15	00:00:00.016	67	36,917
16x16	00:00:00.005	18	6,954
17x17	00:00:00.054	172	144,085
18x18	00:00:06.256	16,229	10,835,714
19x19	00:00:00.072	343	177,304
20x20	00:00:00.072	216	155,813
21x21	00:00:00.092	282	171,238
22x22	00:00:00.089	142	120,023
23x23	00:00:50.231	84,786	67,222,960
24x24	00:00:02.629	4,305	3,829,157
25x25	00:00:01.636	3,268	2,502,639
26x26	00:00:16.392	25,167	20,198,711
27x27	00:01:15.713	92,814	98,743,399
28x28	00:00:09.897	11,008	12,137,938
29x29	00:00:13.392	20,629	22,838,424
30x30	00:00:00.409	679	696,482
31x31	00:00:05.386	10,221	8,993,129
32x32	00:01:49.602	116,283	151,800,684
33x33	00:00:01.136	1,704	2,066,471
34x34	00:00:34.985	40,978	45,436,467
35x35	00:00:12.991	18,472	19,219,173
36x36	00:03:32.730	217,539	265,720,097
37x37	00:00:42.944	56,905	61,576,669
38x38	00:00:00.863	1,660	1,812,527
39x39	01:14:47.299	7,970,705	9,871,119,474
40x40	00:00:35.196	48,956	52,211,662
rotation_test	00:00:00.000	4	377

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**	**Nodes**	**Propagations**
8x8	00:00:00.001	3	150
9x9	00:00:00.001	5	357
10x10	00:00:00.001	5	461
11x11	00:00:00.006	10	1,265
12x12	00:00:00.004	45	8,116
13x13	00:00:00.001	11	2,159
14x14	00:00:00.011	19	2,599
15x15	00:00:00.020	142	27,081
16x16	00:00:00.002	9	1,814
17x17	00:00:00.010	80	24,551
18x18	00:00:00.005	26	5,218
19x19	00:00:00.043	316	91,564
20x20	00:00:00.057	536	123,172
21x21	00:00:00.338	900	304,328
22x22	00:00:00.186	544	190,132
23x23	00:00:07.614	19,260	9,171,901
24x24	00:00:00.516	1,488	638,669
25x25	00:00:04.865	12,490	4,788,188
26x26	00:00:01.409	3,595	2,087,114
27x27	00:00:02.361	4,784	3,049,562
28x28	00:00:05.475	12,410	6,058,207
29x29	00:00:09.348	19,882	12,918,825
30x30	00:00:00.095	287	100,741
31x31	00:00:00.229	916	301,542
32x32	00:00:00.252	387	267,579
33x33	00:00:00.477	1,580	770,610
34x34	00:00:00.593	1,701	827,950
35x35	00:00:00.472	1,752	935,117
36x36	00:00:05.979	14,412	8,047,902
37x37	00:00:52.361	94,388	48,406,756
38x38	00:00:01.842	6,150	2,711,326
39x39	00:04:02.096	375,575	235,754,553
40x40	00:00:03.396	7,408	3,362,652
rotation_test	-	-	-

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**	**Nodes**	**Propagations**
8x8	00:00:00.001	6	704
9x9	00:00:00.001	8	1,385
10x10	00:00:00.001	9	1,861
11x11	00:00:00.008	15	3,815
12x12	00:00:00.016	98	46,406
13x13	00:00:00.007	29	11,036
14x14	00:00:00.005	25	9,711
15x15	00:00:00.037	67	39,702
16x16	00:00:00.006	18	8,008
17x17	00:00:00.104	172	156,596
18x18	00:00:03.629	9,607	6,843,426
19x19	00:00:00.089	343	204,080
20x20	00:00:00.058	216	174,629
21x21	00:00:00.103	282	195,347
22x22	00:00:00.059	142	131,788
23x23	00:00:09.474	17,360	15,664,952
24x24	00:00:02.085	4,242	3,798,154
25x25	00:00:00.915	1,811	1,826,444
26x26	00:00:11.858	18,553	16,727,462
27x27	00:01:16.310	90,225	110,792,895
28x28	00:00:08.492	10,046	13,668,416
29x29	00:00:18.408	23,712	30,221,339
30x30	00:00:01.337	1,558	1,849,202
31x31	00:00:10.415	16,441	16,012,085
32x32	00:02:00.591	124,091	198,341,744
33x33	00:00:01.227	1,804	2,694,263
34x34	00:00:15.616	17,082	22,890,425
35x35	00:00:18.245	20,540	25,953,977
36x36	00:00:13.237	15,788	20,839,347
37x37	00:00:46.591	59,444	69,157,510
38x38	00:00:01.172	1,794	2,453,420
39x39	01:04:52.110	6,450,084	8,859,613,257
40x40	00:00:42.941	48,634	62,681,215
rotation_test	00:00:00.000	4	439

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 con-

straint. 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 Base Model			
Instance	**Time**	**Nodes**	**Propagations**
8x8	00:00:00.000	5	102
9x9	00:00:00.000	6	214
10x10	00:00:00.000	6	271
11x11	00:00:00.000	14	739
12x12	00:00:00.001	13	1,091
13x13	00:00:00.000	13	905
14x14	00:00:00.001	11	736
15x15	00:00:00.001	13	897
16x16	00:00:00.000	12	1,021
17x17	00:00:00.002	35	4,494
18x18	00:00:00.454	2,759	533,562
19x19	00:00:00.001	17	1,954
20x20	00:00:00.045	659	119,592
21x21	00:00:00.001	22	2,618
22x22	00:00:00.108	972	199,771
23x23	00:00:00.965	4,974	1,203,317
24x24	00:00:00.002	24	3,684
25x25	00:00:00.467	3,715	953,430
26x26	00:00:04.947	33,030	12,338,309
27x27	00:00:00.004	42	9,089
28x28	00:00:01.104	4,707	1,339,405
29x29	00:00:02.633	10,540	3,973,251
30x30	00:00:00.005	54	10,476
31x31	00:00:00.001	28	4,340
32x32	00:00:16.341	87,194	36,841,443
33x33	00:00:01.717	9,931	4,226,475
34x34	00:00:00.009	90	18,068
35x35	00:00:00.003	34	6,829
36x36	00:00:00.004	35	7,112
37x37	00:00:42.772	117,956	47,269,882
38x38	00:00:01.838	10,411	1,930,077
39x39	00:00:52.227	146,848	61,008,260
40x40	00:00:00.002	35	5,639
rotation_test	-	-	-

Results Rotation Model			
Instance	**Time**	**Nodes**	**Propagations**
8x8	00:00:00.000	9	152
9x9	00:00:00.001	10	291
10x10	00:00:00.000	12	408
11x11	00:00:00.002	39	2,481
12x12	00:00:00.001	21	1,187
13x13	00:00:00.002	32	2,284
14x14	00:00:00.001	26	1,673
15x15	00:00:00.002	37	2,921
16x16	00:00:00.002	23	1,634
17x17	00:00:00.002	33	3,085
18x18	00:00:01.331	7,459	1,588,000
19x19	00:00:00.115	944	150,992
20x20	00:00:00.021	200	28,185
21x21	00:00:00.010	106	15,156
22x22	00:00:00.017	97	16,673
23x23	00:00:18.932	79,192	24,832,540
24x24	00:00:03.594	21,388	5,648,587
25x25	00:00:02.008	7,423	2,089,151
26x26	00:00:16.716	55,605	16,920,460
27x27	00:00:04.157	17,996	6,116,759
28x28	00:00:11.221	33,631	11,319,838
29x29	00:17:18.251	3,381,737	1,219,343,023
30x30	00:00:52.789	151,180	41,896,201
31x31	00:00:03.975	14,202	3,662,969
32x32	-	-	-
33x33	00:00:00.031	197	41,557
34x34	00:00:05.361	18,808	5,375,826
35x35	00:00:12.866	44,325	13,693,232
36x36	00:00:13.551	44,966	14,045,118
37x37	-	-	-
38x38	00:00:02.048	8,868	2,221,247
39x39	-	-	-
40x40	00:00:22.142	74,395	19,584,365
rotation_test	00:00:00.000	8	161

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 [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:

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

  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:

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

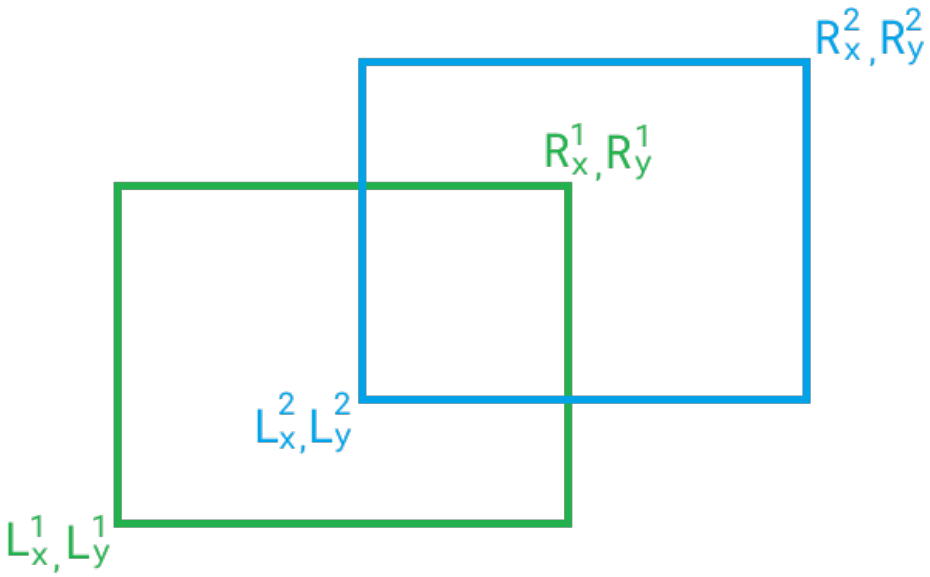

Figure 3.1: Overlapping Model

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

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

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

$$= Width$$

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

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

$$= Height$$

Results				
Instance	**Time**	**Nodes**	**Propagations**	**Memory**
8x8	00:00:00.024	7	196	87,820
9x9	00:00:00.024	15	335	87,820
10x10	00:00:00.018	44	954	4,250
11x11	00:00:00.040	148	5,707	4,860
12x12	00:00:00.039	214	5,214	5,210
13x13	00:00:00.041	54	1,119	5,370
14x14	00:00:00.043	96	2,641	5,730
15x15	00:00:00.066	507	5,574	6,370
16x16	00:00:00.105	388	22,306	7,270
17x17	00:00:00.193	781	40,421	8,500
18x18	00:00:00.984	5,922	449,119	16,730
19x19	00:00:00.443	3,356	143,315	11,960
20x20	00:00:00.504	2,378	199,929	13,330
21x21	00:00:00.729	5,095	348,298	16,470
22x22	00:00:00.446	3,668	209,723	15,210
23x23	00:00:00.984	7,436	439,412	21,530
24x24	00:00:01.172	7,531	527,852	22,110
25x25	00:00:02.296	10,267	906,606	29,400
26x26	00:00:18.432	23,295	2,385,900	56,840
27x27	00:00:03.940	14,291	1,122,428	36,770
28x28	00:00:37.466	32,367	5,398,044	82,150
29x29	00:00:44.512	48,695	7,833,701	80,210
30x30	00:00:06.008	6,134	452,326	44,070
31x31	00:00:01.404	5,586	477,150	44,070
32x32	00:02:12.063	124,685	28,828,494	130,759
33x33	00:00:34.455	34,530	4,603,388	70,410
34x34	00:00:07.254	10,249	925,691	82,150
35x35	00:00:24.671	28,195	3,962,940	87,820
36x36	00:00:27.781	11,203	1,273,816	91,330
37x37	00:05:05.860	198,820	62,031,017	211,940
38x38	00:00:03.388	7,515	1,216,248	87,820
39x39	00:03:12.324	172,697	45,713,826	189,550
40x40	00:00:02.630	6,383	859,687	87,820
rotation_test	-	-	-	-

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":

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

Results				
Instance	**Time**	**Nodes**	**Propagations**	**Memory**
8x8	00:00:00.021	52	140	211,940
9x9	00:00:00.019	103	471	211,940
10x10	00:00:00.021	374	607	87,820
11x11	00:00:00.041	142	2,295	87,820
12x12	00:00:00.084	487	17,402	87,820
13x13	00:00:00.095	257	26,864	87,820
14x14	00:00:00.100	362	21,240	87,820
15x15	00:00:00.159	525	57,603	87,820
16x16	00:00:00.087	399	12,846	87,820
17x17	00:00:00.371	1,088	165,078	87,820
18x18	00:00:25.060	20,254	4,655,919	87,820
19x19	00:00:01.458	3,160	684,733	87,820
20x20	00:00:00.651	3,022	313,670	87,820
21x21	00:00:01.386	4,354	629,898	87,820
22x22	00:00:00.371	1,802	98,081	87,820
23x23	00:00:23.778	17,883	3,802,390	87,820
24x24	00:00:03.223	9,188	969,847	87,820
25x25	00:00:08.041	6,649	1,113,972	87,820
26x26	00:00:24.968	22,894	4,452,757	130,759
27x27	00:00:10.064	14,148	2,083,453	87,820
28x28	00:00:08.320	9,009	1,132,277	87,820
29x29	00:00:33.789	24,402	3,906,977	88,600
30x30	00:00:13.749	13,058	1,021,942	130,759
31x31	00:00:11.827	4,628	500,544	130,759
32x32	00:05:20.900	162,248	72,507,269	228,930
33x33	00:00:57.685	48,708	13,329,045	88,770
34x34	00:00:15.569	12,762	1,623,818	88,770
35x35	00:00:24.271	21,457	3,966,717	189,550
36x36	00:00:42.454	30,754	7,084,316	91,320
37x37	00:02:58.149	121,821	38,672,489	189,550
38x38	00:00:19.120	17,460	2,821,474	91,320
39x39	00:09:31.279	202,553	55,241,230	193,430
40x40	00:00:08.420	11,007	1,357,752	211,940
rotation_test	-	-	-	-

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**	**Nodes**	**Propagations**	**Memory**
8x8	-	-	-	-
9x9	-	-	-	-
10x10	00:00:00.144	721	26,685	211,940
11x11	00:00:00.067	579	5,680	211,940
12x12	00:00:01.408	5,214	443,395	211,940
13x13	00:00:00.448	2,119	140,276	211,940
14x14	00:00:02.592	8,111	756,747	211,940
15x15	00:00:02.032	7,798	683,163	211,940
16x16	00:00:03.967	11,880	1,238,221	211,940
17x17	00:00:26.176	5,651	364,315	211,940
18x18	00:00:28.966	18,584	1,035,514	211,940
19x19	00:00:32.796	22,376	2,354,741	211,940
20x20	00:00:29.592	16,027	1,459,495	211,940
21x21	00:00:46.402	39,691	8,637,344	189,550
22x22	00:00:35.597	25,020	3,011,708	211,940
23x23	00:00:36.571	37,053	2,927,184	189,550
24x24	00:01:18.530	83,289	15,993,390	228,930
25x25	00:02:36.290	140,045	25,825,689	211,940
26x26	00:05:18.520	266,107	58,347,287	189,550
27x27	00:04:49.714	248,500	62,248,419	228,930
28x28	00:03:14.506	175,627	37,562,461	127,420
29x29	00:18:32.908	564,825	144,571,773	320,540
30x30	00:02:23.401	141,093	29,682,469	228,930
31x31	00:00:53.787	55,579	7,421,610	228,930
32x32	-	-	-	-
33x33	00:04:09.668	218,937	46,436,633	320,540
34x34	00:00:36.893	24,340	4,007,587	320,540
35x35	00:03:51.419	182,898	42,674,674	320,540
36x36	00:05:13.999	222,110	57,051,458	320,540
37x37	-	-	-	-
38x38	-	-	-	-
39x39	-	-	-	-
40x40	-	-	-	-
rotation_test	-	-	-	-

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**	**Nodes**	**Propagations**	**Memory**
8x8	-	-	-	-
9x9	-	-	-	-
10x10	-	-	-	-
11x11	-	-	-	-
12x12	-	-	-	-
13x13	-	-	-	-
14x14	-	-	-	-
15x15	-	-	-	-
16x16	-	-	-	-
17x17	-	-	-	-
18x18	-	-	-	-
19x19	-	-	-	-
20x20	-	-	-	-
21x21	-	-	-	-
22x22	-	-	-	-
23x23	-	-	-	-
24x24	-	-	-	-
25x25	-	-	-	-
26x26	-	-	-	-
27x27	-	-	-	-
28x28	-	-	-	-
29x29	-	-	-	-
30x30	-	-	-	-
31x31	-	-	-	-
32x32	-	-	-	-
33x33	-	-	-	-
34x34	-	-	-	-
35x35	-	-	-	-
36x36	-	-	-	-
37x37	-	-	-	-
38x38	-	-	-	-
39x39	-	-	-	-
40x40	-	-	-	-
rotation_test	-	-	-	-

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**	**Nodes**	**Propagations**	**Memory**
8x8	-	-	-	-
9x9	-	-	-	-
10x10	-	-	-	-
11x11	-	-	-	-
12x12	-	-	-	-
13x13	-	-	-	-
14x14	-	-	-	-
15x15	-	-	-	-
16x16	-	-	-	-
17x17	-	-	-	-
18x18	-	-	-	-
19x19	-	-	-	-
20x20	-	-	-	-
21x21	-	-	-	-
22x22	-	-	-	-
23x23	-	-	-	-
24x24	-	-	-	-
25x25	-	-	-	-
26x26	-	-	-	-
27x27	-	-	-	-
28x28	-	-	-	-
29x29	-	-	-	-
30x30	-	-	-	-
31x31	-	-	-	-
32x32	-	-	-	-
33x33	-	-	-	-
34x34	-	-	-	-
35x35	-	-	-	-
36x36	-	-	-	-
37x37	-	-	-	-
38x38	-	-	-	-
39x39	-	-	-	-
40x40	-	-	-	-
rotation_test	-	-	-	-

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**	**Nodes**	**Propagations**	**Memory**
8x8	-	-	-	-
9x9	-	-	-	-
10x10	-	-	-	-
11x11	-	-	-	-
12x12	-	-	-	-
13x13	-	-	-	-
14x14	-	-	-	-
15x15	-	-	-	-
16x16	-	-	-	-
17x17	-	-	-	-
18x18	-	-	-	-
19x19	-	-	-	-
20x20	-	-	-	-
21x21	-	-	-	-
22x22	-	-	-	-
23x23	-	-	-	-
24x24	-	-	-	-
25x25	-	-	-	-
26x26	-	-	-	-
27x27	-	-	-	-
28x28	-	-	-	-
29x29	-	-	-	-
30x30	-	-	-	-
31x31	-	-	-	-
32x32	-	-	-	-
33x33	-	-	-	-
34x34	-	-	-	-
35x35	-	-	-	-
36x36	-	-	-	-
37x37	-	-	-	-
38x38	-	-	-	-
39x39	-	-	-	-
40x40	-	-	-	-
rotation_test	-	-	-	-

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*. Let's define the following predicate, where p is the index of the current present:

$$correct\_dimension(p, dx, dy) \leftrightarrow \bigvee_{\substack{x_0 \in [1, Width-dx] \\ y_0 \in [1, Height-dy]}} \left(\bigwedge_{\substack{x \in [x_0, x_0+dx] \\ y \in [y_0, y_0+dy]}} Paper[x, y, p] \right) \vee \left(\bigwedge_{\substack{x \in [1, x_0] \cup [x_0+dx+1, Width] \\ y \in [1, y_0] \cup [y_0+dy+1, Height]}} \neg Paper[x, y, p] \right)$$

  So we end up with the full constrain:

$$\bigwedge_{p \in [1, Presents]} correct\_dimension(p, Dimension_x[p], Dimension_y[p])$$

- ***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):

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

  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:

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

  Cols:

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

Results				
Instance	**Time**	**Nodes**	**Propagations**	**Memory**
8x8	-	-	-	-
9x9	-	-	-	-
10x10	-	-	-	-
11x11	-	-	-	-
12x12	-	-	-	-
13x13	-	-	-	-
14x14	-	-	-	-
15x15	-	-	-	-
16x16	-	-	-	-
17x17	-	-	-	-
18x18	-	-	-	-
19x19	-	-	-	-
20x20	-	-	-	-
21x21	-	-	-	-
22x22	-	-	-	-
23x23	-	-	-	-
24x24	-	-	-	-
25x25	-	-	-	-
26x26	-	-	-	-
27x27	-	-	-	-
28x28	-	-	-	-
29x29	-	-	-	-
30x30	-	-	-	-
31x31	-	-	-	-
32x32	-	-	-	-
33x33	-	-	-	-
34x34	-	-	-	-
35x35	-	-	-	-
36x36	-	-	-	-
37x37	-	-	-	-
38x38	-	-	-	-
39x39	-	-	-	-
40x40	-	-	-	-
rotation_test	-	-	-	-

4.2 Rotation Model

As for **CP** and **SMT**, we just need another variable that keeps track of the rotation of each presnt in the paper sheet:

In this case, we do not need to use a proxy to gather the correct dimension, we just check the correct dimension in two different ways: the normal or the rotated one. Like this, we can place each present in the normal *OR* the rotated

way and this is the resulting constrain:

$$\bigwedge_{p \in [1, Presents]} (\\ correct\_dimension(p, Dimension_x[p], Dimension_y[p]) \vee \\ correct\_dimension(p, Dimension_y[p], Dimension_x[p]) \\ )$$

As we can see, by switching the two dimension, we can simply rotate the present.

Results				
Instance	**Time**	**Nodes**	**Propagations**	**Memory**
8x8	-	-	-	-
9x9	-	-	-	-
10x10	-	-	-	-
11x11	-	-	-	-
12x12	-	-	-	-
13x13	-	-	-	-
14x14	-	-	-	-
15x15	-	-	-	-
16x16	-	-	-	-
17x17	-	-	-	-
18x18	-	-	-	-
19x19	-	-	-	-
20x20	-	-	-	-
21x21	-	-	-	-
22x22	-	-	-	-
23x23	-	-	-	-
24x24	-	-	-	-
25x25	-	-	-	-
26x26	-	-	-	-
27x27	-	-	-	-
28x28	-	-	-	-
29x29	-	-	-	-
30x30	-	-	-	-
31x31	-	-	-	-
32x32	-	-	-	-
33x33	-	-	-	-
34x34	-	-	-	-
35x35	-	-	-	-
36x36	-	-	-	-
37x37	-	-	-	-
38x38	-	-	-	-
39x39	-	-	-	-
40x40	-	-	-	-
rotation_test	-	-	-	-

4.3 On the hardness of SAT modelling

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 strugled to achieve the implementation of new models, but we have been discouraged by the loss of performance in the **SMT Symmetry Breaking** models, so we decided to try to improve as best

as possible the basical models.
Another weak point for the **SAT** is that we can't write a general purpose program in **SMT2Lib** standard. Indeed, to achieve such a generalization, we need to use numerical calculus, not completely available in the **Propositional Logic**.

4.4 Results

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/`