The 2D Strip Packing Problem - VLSI Design

Alessandro D'Amico (alessandro.damico5@studio.unibo.it)
Andrea Virgillito (andrea.virgillito2@studio.unibo.it)
Sfarzo El Husseini (sfarzo.elhusseini@studio.unibo.it)

1 Introduction

1.1 Description of the problem

The assignment given consists in solving the following problem: placing a set of chips[1] in a way to occupy the least possible space on a silicon plate whose shape is rectangular and whose width is fixed. Occupying the least possible space consists therefore in minimizing the height of the silicon plate. This problem is commonly known in the literature as *2D strip packing problem*.
In this document we describe a solution implemented with four optimization techniques: *CP*,*SAT*,*SMT* and *MIP* (in this last case, the model contains just integer variables in linear constraints, therefore, it can be considered an *ILP* model).

1.2 Work management

Alessandro focused on the production and the benchmarking of *SAT* and *LP* models and on the additional benchmarking of *CP* on *OR-Tools CP Solver*, as well as the whole refactoring of the project. Sfarzo focused on the production and benchmarking of the SMT model and Andrea focused on the production and benchmarking of the CP model.
The work has been carried out in more than one month of work, whose, beyond the time spent reading papers and writing models included:

- **Refactoring of all the repository** in a modular and homogeneous way (so that everyone could use the same "framework" for all the solving methods)
- **Porting of all the methods to a unique python framework** with which the user can easily verify the work done and interact with the model pipelines
- **Refactoring of the MIP models** that were initially ported through python and minizinc (which was having a direct access to the installed solvers: *CPLEX* and *Gurobi*). The refactored models have been translated in the *Google OR-Tools framework*, which has shown to be very easy to use, despite its lower flexibility if compared to custom languages such as *AMPL* (in fact, some solvers such as *CPLEX* require the re-compilation of the whole $OR-Tools$ framework to be used). Using AMPL would have required to learn a new language from scratch, but also a porting with Python, to input the height upper bound with our common method, and to obtain the graphical solution with the corresponding elapsed times.
- **Gurobi install**: it is possible to use even other solvers to solve MIP problems, but Gurobi has shown to be much better than the open source solvers and other commercial solvers such as CPLEX. Independently from the platform in use, the academic licence can be obtained just through the official university network.

1.3 Benchmark setup

All the benchmarks were realized using the machine *Xiaomi Air 13 (2017)*, having the following specifications:

[1] The widths and the lengths of the chips, having rectangular shape, are a priori known.

- OS: Microsoft Windows 10 Home (x64)
- CPU: Intel(R) Core(TM) i5-7200U @2.50GHz, 2 physical cores, 4 virtual cores
- RAM: 8.00 GB DDR4 @2133MHz
- Storage: 256 GB SSD SAMSUNG MZVLW256HEHP-00000

1.4 Common techniques and variables

In the various models, we've shared some input constants and variables that are described as follows:

1.4.1 Base model

Each instance of this problem is characterized by:

- W : constant, it's the fixed width of the rectangular silicon plate. It's an integer number.
- H : variable, it's the height of the rectangular silicon plate to be minimized. It is an integer number so that $H \in \{H_{LB}, .., H_{UB}\}$
  - H_{LB} has been computed considering a perfect packaging of all the rectangular chips inside the silicon rectangle: in that case the occupied area A is such that $A = \sum_{i \in n_{rectangles}} height_i \cdot width_i$ and $H_{LB} = \frac{A}{W}$
  - H_{UB} has been computed applying the *Bottom-Up Left-Justified (BL)* technique described by *Baker et al.*[1][2]: The rectangles are placed on the lowest available spot on the left, starting from the left bottom corner. In addition to this technique, the rectangles were ordered by width size. The BL technique is easy to implement and in the most complex cases (such as instance 40) it is computed in less than 5 seconds.
    In case of rotations it is sufficient to use $H_{UB_{rotation}}$ instead of H_{UB}, which, in addition to the previous, will consider also the sum of the shortest side (This is useful just in the cases in which the main silicon plate is tight if compared to the average dimension of the rectangular chips). In practice: $H_{UB_{rotation}} = min(H_{UB}, \sum_{i \in n_{rectangles}} \min(width_i, height_i))$
- $n_{rectangles}$: constant, it is the number of rectangular chips to be placed inside the silicon plate
- $width_i \quad i \in \{0, .., n_{rectangles} - 1\}$: constant, it is the width of the rectangle i.
- $height_i \quad i \in \{0, .., n_{rectangles} - 1\}$: constant, it is the height of the rectangle i.
- $x_i \quad i \in \{0, .., n_{rectangles} - 1\}$: variable, it's the horizontal coordinate of the low-bottom corner of each rectangle with respect to the low-bottom corner of the silicon plate. $x_i \in \{0, .., W - width_i\}$
- $y_i \quad i \in \{0, .., n_{rectangles} - 1\}$: variable, it's the horizontal coordinate of the low-bottom corner of each rectangle with respect to the low-bottom corner of the silicon plate. $y_i \in \{0, .., H - height_i\}$

1.4.2 Rotation model

- $r_i \quad i \in \{0, .., n_{rectangles} - 1\}$: variable, it's the rotation of each rectangle, in the variant of the problem in which this is allowed[3]. being r_i binary variables, we have $r_i \in \{0, 1\}$ or equivalently $r_i \in \{False, True\}$ ($r_i \leftarrow True$ if the rectangle has been rotated in the final placement, $r_i \leftarrow False$ otherwise).

[2]The BL technique guarantees an Upper Bound that is in the worst case $3 \cdot OPT(I)$, where $OPT(I)$ is the best possible height for the collection of rectangular chips I

[3]In the final output *.txt* solutions, the rotation parameter is not specified, instead, the height and width are directly swapped

1.4.3 Implied and Symmetry Breaking constraints

These additional constraints are mostly derived from *Soh et Al.*[2]. It is worth highlighting that the first two kinds of constraints described below (*LR* and *SR*) are rarely applied, therefore in some instances it is possible that they have no effect.
On the other hand, the *LS* constraint is always applied and, even though in some methods increases the search space exploration speed, sometimes causes the cutting of "closer" alternative solutions.

- **Large Rectangles (LR) Implied constraint**
  If the sum of two rectangles widths is bigger than the width of the strip, i.e. $width_i + width_j > W$ (Figure 1), we can assume that they can't be placed one on the left or right side of each other.

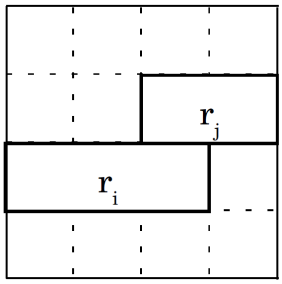

Figure 1: The rectangles r_i and r_j are too wide to be placed one on the side of each other. Image taken from *Soh et Al.*[2].

  We can also apply the same reasoning to the height, because if $height_i + height_j > H$ we can assume that those rectangles can't be placed one above or under each other.

- **Same Sized Rectangles (SR) Symmetry Breaking constraint**
  If two rectangles are such that $width_i = width_j$ and $height_i = height_j$ (Figure 2), we can choose to have rectangle i left-under j.

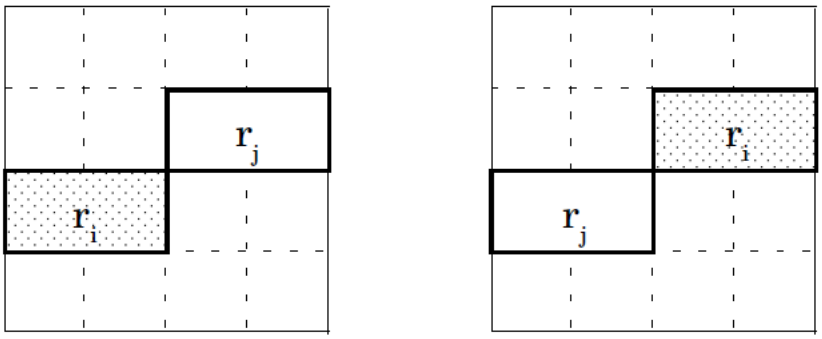

Figure 2: The rectangles r_i and r_j are identical, swapping them leads us to the same kind of solution. Image taken from *Soh et Al.*[2].

- **Largest Sized Rectangle (LS) Symmetry Breaking constraint**
  We can assume to have the *largest* rectangle (in terms of area) in the left bottom half of the strip. This makes it possible to restrict the domain with respect to the reflective symmetries of the largest rectangle (Figure 3):

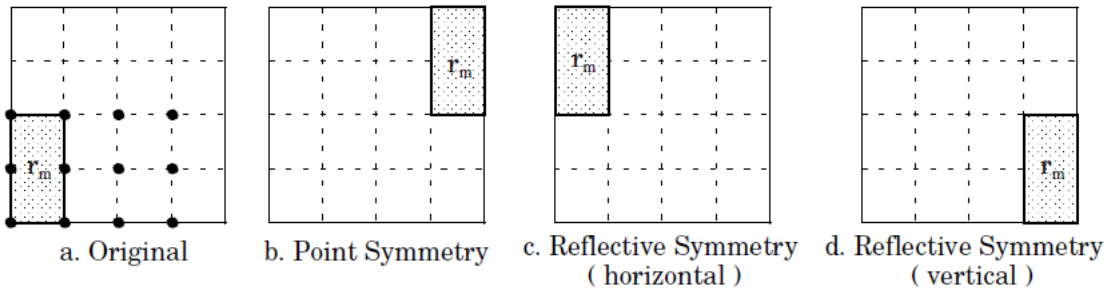

Figure 3: For each rectangle we have reflective symmetries with respect to the final silicon plate cut. We cannot break this kind of symmetry for each rectangular chip, but we can do that for the rectangle having the biggest area between them. Image taken from *Soh et Al.*[2].

- **Squares Symmetry Breaking constraint (just for models with rotations)** In the version with rotations it is useless to rotate squares, therefore if $width_i = height_i$: the additional variable r_i is removed from the constraints or set to *False*.

2 CP Model

2.1 Decision variables

The decision variables correspond to the ones described in Section 1.4. Moreover, we've added

- an array of variables x_{coords} grouping all the x coordinates of the rectangle
- an array of variables y_{coords} grouping all the y coordinates of the rectangle
- an array of variables $widths$ grouping all the $width$ coordinates of the rectangle
- an array of variables $heights$ grouping all the $height$ coordinates of the rectangle

2.2 Objective function

Our goal is to minimize the height, therefore the objective function is:

$$minimize(H)$$

In addition to this, we adopt a sequential search (in Minizinc it is called *seq_search*) that as first step tries the height value in increasing order (using the option *indomain_min*) and as second step tries to place the rectangles[4] that, because of the computation for the height upper bound H_{UB} described in section 1.4, are already ordered by width.

2.3 Constraints

To model the problem we've used just 4 main constraints:

- $diffn(x_{coords}, y_{coords}, widths, heights)$: this global constraint avoids the overlapping between rectangles, taking as first and second parameters the coordinates on the x and y axis of the left-bottom corner of each of those rectangles and as third and fourth parameter the widths and heights.
- $cumulative(y_{coords}, heights, widths, W)$: this global constraint is not mandatory to model the problem, but is useful to tackle it from a different perspective (as if it was a scheduling problem). This constraint takes as a first parameter the start times of a set of tasks, as second parameter the duration of those tasks, as third parameter the resource requirement for each task and as fourth parameter the overall bound for such resources, which has to be respected during the duration of all those tasks. In our case, we can think of the coordinates on the y axis of the left-bottom corner of each rectangle being the start times and the heights of the rectangles being the durations of those. The widths can be interpreted as the resources, with W being the overall bound not to be exceed. We've experimented the same constraint using as parameters $x_{coords}, widths, heights$ and W, but this didn't provide further improvements.
- $x_i + widths_i \leq W \qquad \forall i \in \{0, .., n_{rectangles} - 1\}$: This constraint ensures that each rectangle doesn't exceed the prescribed fixed width W of the silicon plate.
- $y_i + height_i \leq H \qquad \forall i \in \{0, .., n_{rectangles} - 1\}$: This constraint ensures that each rectangle doesn't exceed the height H of the silicon plate.

2.3.1 Implied constraints

Large Rectangles (LR) Implied constraint If the sum of two rectangles widths is bigger than the width of the strip (i.e. $width_i + width_j > W$) we can assume that they can't be placed one on the left or right side of the other (as explained in Section 1.4.3) and we can represent this constraint as follows:

$$(width_i + width_j > W) \implies |y_i - y_j| > min(height_i, height_j)$$
$$\forall j > i, \qquad i, j \in \{0, .., n_{rectangles} - 1\}$$

[4] trying values for the x_i and y_i for each rectangle i

In this way, we constraint one of the two rectangles to be in a position that is higher or lower than the other of at least the smallest height between them.
N.B.: we didn't use this implied constraint during the benchmark, because it wasn't giving us any improvement (but we've implemented it anyways).

2.3.2 Symmetry breaking constraints

We've applied 2 kinds of Symmetry Breaking Constraints:

Same Sized Rectangles (SR) Symmetry Breaking If two rectangles are such that $width_i = width_j$ and $height_i = height_j$ we can choose to have rectangle i left-under j (as explained in Section 1.4.3):

$$((width_i = width_j) \bigwedge (height_i = height_j)) \implies (x_i \leq x_j \bigwedge y_i \leq y_j)$$
$$\forall j > i, \qquad i, j \in \{0, .., n_{rectangles} - 1\}$$

Largest Sized Rectangle (LS) Symmetry Breaking We can also assume to have the *largest* rectangle (in terms of area) in the left bottom half of the strip, in order to avoid reflexive symmetries (as explained in Section 1.4.3). With *largest*, we mean the index of the rectangle having the biggest are among the ones in the same instance, so

$$largest = \{i \in 0, .., n_{rectangles} - 1 | height_i \cdot width_i \geq height_j \cdot width_j \quad \forall j \in \{0, .., n_{rectangles} - 1\}$$

$$(x_{largest} \leq \frac{W - width_{largest}}{2}) \bigwedge (y_{largest} \leq \frac{H - height_{largest}}{2})$$

In this way we constraint the largest rectangle to have its low bottom corner placed on the low bottom region of the whole silicon rectangular plate.

2.4 Rotation

To describe rotation we used the variables r_i already described in Section 1.4.2:

$$(r_i \bigwedge width_{i_{new}} = height_i \bigwedge height_{i_{new}} = width_i) \bigoplus (\neg r_i \bigwedge width_{i_{new}} = width_i \bigwedge height_{i_{new}} = height_i)$$
$$\forall i, \qquad i \in \{0, .., n_{rectangles} - 1\}$$

If the rectangle is rotated, width and height are swapped, otherwise, they're kept as they are.

2.4.1 Symmetry Breaking

Squares rotation is of course useless (as described in Section 1.4), therefore we posed a constraint which avoids rotations for those:

$$height_i = width_i \implies r_i = False \qquad \forall i \in \{0, .., n_{components} - 1\}$$

2.5 Validation

2.5.1 Experimental design

We've used the Minizinc Python API to make the porting of the parsed instances parameters and of the H_{UB} computation, while the models are in the *.mzn* files. The best results have been obtained enabling parallel solving (through the *-p* flag) and selecting the *free-search* option (through the *-f* flag) on Chuffed and Or-Tools: with this last setting, the solver is not required to ignore the annotations, but it is allowed to do so. The device used to benchmark this method is described in Section 1.3.

2.5.2 Experimental results

The results obtained with the solvers *Chuffed* and *Or-Tools* in the 4 different variants are described by the table 1 (reporting the best value obtained for H for each instance) and by the graphs in Figure 4 and Figure 5 (reporting the elapsed time for each instance).

ins.	Chuffed	OT	Chuffed(SB)	OT(SB)	Chuffed(rot)	OT(rot)	Chuffed(rot-SB)	OT(rot-SB)
1	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**
2	**9**	**9**	**9**	**9**	**9**	**9**	**9**	**9**
3	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**
4	**11**	**11**	**11**	**11**	**11**	**11**	**11**	**11**
5	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**
6	**13**	**13**	**13**	**13**	**13**	**13**	**13**	**13**
7	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**
8	**15**	**15**	**15**	**15**	**15**	**15**	**15**	**15**
9	**16**	**16**	**16**	**16**	**16**	**16**	**16**	**16**
10	**17**	**17**	**17**	**17**	**17**	**17**	**17**	**17**
11	**18**	**18**	**18**	**18**	**18**	**18**	**18**	**18**
12	**19**	**19**	**19**	**19**	**19**	**19**	**19**	**19**
13	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**
14	**21**	**21**	**21**	**21**	**21**	**21**	**21**	**21**
15	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**
16	**23**	**23**	**23**	**23**	**23**	**23**	**23**	**23**
17	**24**	**24**	**24**	**24**	**24**	**24**	**24**	**24**
18	**25**	**25**	**25**	**25**	**25**	**25**	**25**	**25**
19	**26**	**26**	**26**	**26**	**26**	**26**	**26**	**26**
20	**27**	**27**	**27**	**27**	**27**	**27**	**27**	**27**
21	**28**	**28**	**28**	**28**	**28**	**28**	**28**	**28**
22	**29**	**29**	**29**	**29**	**29**	**29**	- (38**)	**29**
23	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**
24	**31**	**31**	**31**	**31**	**31**	**31**	**31**	**31**
25	**32**	**32**	**32**	**32**	- (40**)	**32**	- (40**)	**32**
26	**33**	**33**	**33**	**33**	**33**	**33**	**33**	**33**
27	**34**	**34**	**34**	**34**	**34**	**34**	**34**	**34**
28	**35**	**35**	**35**	**35**	**35**	**35**	**35**	**35**
29	**36**	**36**	**36**	**36**	**36**	**36**	**36**	**36**
30	**37**	**37**	**37**	38	- (44**)	38	- (44**)	38
31	**38**	**38**	**38**	**38**	**38**	**38**	**38**	**38**
32	- (43**)	**39**	**39**	40	- (43**)	40	- (43**)	40
33	**40**	**40**	**40**	**40**	**40**	**40**	**40**	**40**
34	**40**	**40**	**40**	**40**	**40**	**40**	**40**	**40**
35	**40**	**40**	**40**	**40**	**40**	**40**	**40**	**40**
36	**40**	**40**	**40**	**40**	**40**	**40**	**40**	**40**
37	**60**	61	- (73**)	61	- (73**)	61	- (73**)	61
38	**60**	61	**60**	61	- (74**)	61	- (74**)	61
39	**60**	**60**	**60**	**60**	- (69**)	61	- (69**)	61
40	- (103**)	93	- (103**)	92	- (103**)	98	- (103**)	98

Table 1: Table with the best bounds obtained from the CP solvers *Chuffed* and *OR-Tools* (that is marked as *OT* to better visualize the table). Results between parentheses and marked with asterisks are instances for which no additional solution has been computed more than the one already computed through the *Bottom-Left-Justified* method to estimate the upper bound, described in section 1.4. The first two columns correspond to the base models, while the ones marked with *SB* correspond to the version with Symmetry Breaking techniques, the ones marked with *rot* correspond to the models were rotation is enabled and *rot-SB* marks the ones with rotations and squares symmetry breaking enabled

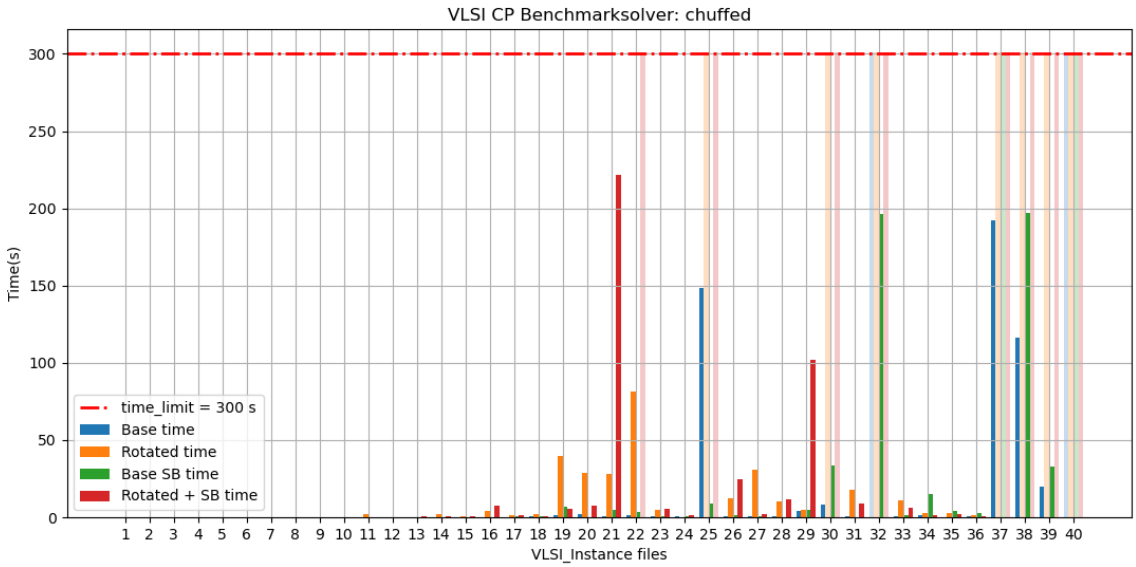

Figure 4: Graph representing the elapsed times during the solving process, using the solver *Chuffed*. For each instance solved to optimality the bars are colored, while the other bars (not solved to optimality or not solved at all) are grey.

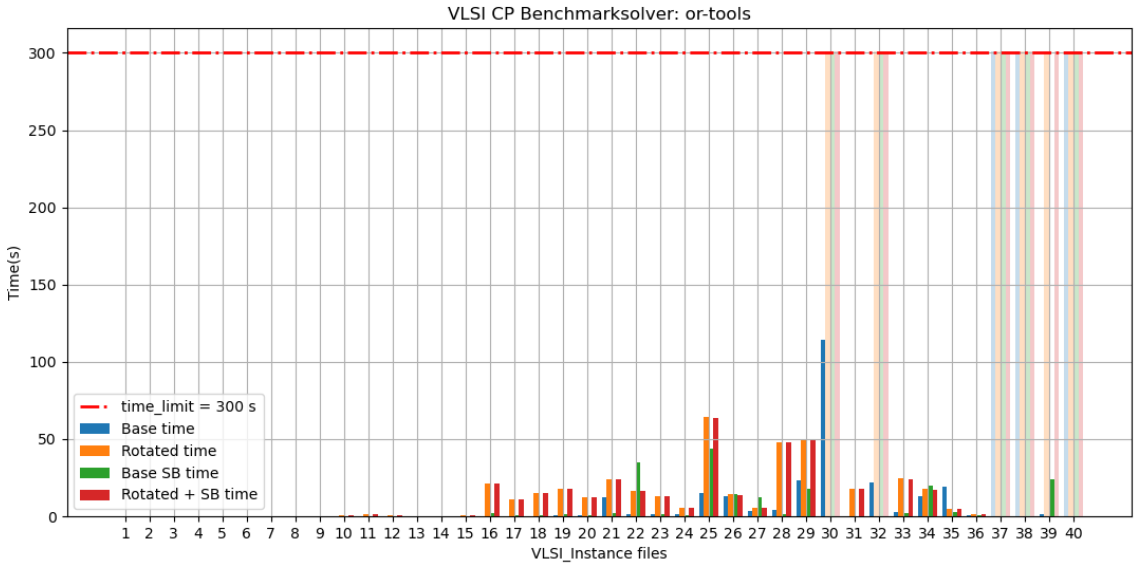

Figure 5: Graph representing the elapsed times during the solving process, using the solver *Or-Tools*. For each instance solved to optimality the bars are colored, while the other bars (not solved to optimality or not solved at all) are grey.

3 SAT Model

3.1 Decision variables

The decision variables used are the same described in Section 1.4 and the corresponding domains were restricted directly manipulating their binary encoding. We've chosen **order encoding** to represent natural numbers as a boolean sequence: this is less obvious than the usual direct encoding, but, as we will see, will make the comparisons easier.
An example of order encoding having 4 as the biggest representable natural number:

$$\begin{aligned}
0 \rightarrow 1111 \rightarrow & enc_0 = TTTT \\
1 \rightarrow 0111 \rightarrow & enc_1 = FTTT \\
2 \rightarrow 0011 \rightarrow & enc_2 = FFTT \\
3 \rightarrow 0001 \rightarrow & enc_3 = FFFT \\
4 \rightarrow 0000 \rightarrow & enc_4 = FFFF
\end{aligned}$$

Using this encoding it is easy to see that the encoding of a natural number $n \in [0, n_{max}]$ is

$$[\underbrace{F..F}_{\text{n}} \ \underbrace{T..T}_{n_{max}\text{-n}}]$$

Using the 0-based indexing for the elements of the encoding enc_a of a natural number a, we have that

$$enc_{a,i} \rightarrow enc_{a,i+1}, \qquad i \in \{0, n_{max} - 2\}$$

rewritten as a pure CNF clause:

$$\neg enc_{a,i} \vee enc_{a,i+1}, \qquad i \in \{0, n_{max} - 2\}$$

It is also worth noticing that

$$enc_{a,j} \Leftrightarrow a \leq j, \qquad j \in \{0, n_{max} - 1\}$$

For each rectangle r_i, the length of the encoding of the x coordinate of the i-th rectangle is

$$|enc_{x_i} = F| = W$$

And the length of the encoding of the y coordinate of the i-th rectangle is

$$|enc_{y_i} = F| = H$$

And to the describe the relative position of the rectangles in the strip we add two kinds of variables:

- the left-right $lr_{i,j}$ which is true if rectangle i is at the left of rectangle j (and therefore j is at the right of i)
- the down-up $du_{i,j}$ which is true if rectangle i is below j (and therefore j is over i).
  N.B.: we've used the variables du, they're symmetrical to the original definition given by *Soh et Al.*[2].

3.2 Objective function

Even in this case, the objective is to maximize the height H, but while in some methods such as *CP* and *MIP* the height H can be considered a variable, in this case H has to be a constant: the rectangle placements will be attempted after fixing the dimensions of the silicon Plate (i.e. W and H will be

assigned before starting the solving process). Known H_{LB} and H_{UB}, the *bisection method* will be used, so the height of the first attempt will be

$$H_{attempt_0} = \lfloor \frac{H_{LB} + H_{UB}}{2} \rfloor$$

and if an assignment for the variables is found, then the new attempted height will be

$$H_{attempt_1} = \lfloor \frac{H_{LB} + H_{attempt_0}}{2} \rfloor$$

, otherwise (in case of failure) it will be

$$H_{attempt_1} = \lfloor \frac{H_{attempt_0} + H_{UB}}{2} \rfloor$$

Instead of using the bisection method we could have started from the actual lower bound H_{LB} and then increment it up to reaching the upper bound, but this is a good method just if we use the strong hypothesis that the given instances can be placed with a height near to H_{LB}.

3.3 Constraints

We should take into account that **all** the whole rectangle and not just the left-bottom corner must fit into the width W and height H of the strip, therefore the following constraints must be satisfied (for each rectangle r_i):

$$enc_{x_i, W-width_i} \wedge ... \wedge enc_{x_i, W-1}$$

$$enc_{y_i, H-height_i} \wedge ... \wedge enc_{y_i, H-1}$$

Then we add the order encoding constraint described in the previous section (for each rectangle r_i):

$$\neg enc_{x_i,j} \vee enc_{x_i,j+1} \qquad j \in \{0, .., W-2\}$$

$$\neg enc_{y_i,k} \vee enc_{y_i,k+1} \qquad k \in \{0, .., H-2\}$$

Finally, we have the **no-overlap** constraints:
Every rectangle has a relative position w.r.t. another, so (for each rectangle r_i,r_j with $i < j$ and $i, j \in \{0, N-1\}$):

$$lr_{i,j} \vee lr_{j,i} \vee du_{i,j} \vee du_{j,i} \tag{1}$$

We should now chain the meaning of the relative position variables with all the other variables and to do that we can think about that, as an example, if a rectangle is at the left of another, the left rectangle left-bottom corner has a distance from the bottom-left corner of the right rectangle that is at least equal to the width of the left rectangle (we can then extend this reasoning to the up-down positioning)

$$lr_{i,j} \Rightarrow x_i + width_i \leq x_j \tag{2}$$

$$lr_{j,i} \Rightarrow x_j + width_j \leq x_i \tag{3}$$

$$du_{i,j} \Rightarrow y_i + height_i \leq y_j \tag{4}$$

$$du_{j,i} \Rightarrow y_j + height_j \leq y_i \tag{5}$$

Encoding of inequalities in SAT If we have an inequality of the form

$$x + c \leq y$$

where $c \in \{0..n_{max}\}$ and so that $|enc_x| = |enc_y| = n_{max}$, then

$$x \leq z, \qquad z \in \{n_{max} - c, .., n_{max}\}$$

$$y \geq t, \qquad t \in \{0, .., c\}$$

which can be rewritten as

$$x \leq z, \qquad z \in \{n_{max} - c, .., n_{max}\}$$

$$\neg(y \leq k), \qquad k \in \{0, .., c-1\}$$

$$(y \leq c + s) \rightarrow (x \leq s), \qquad s \in \{0, .., n_{max} - c - 1\}$$

The formulas above are rewritable as literals, each one corresponding to a simple inequality:

$$\bigwedge enc_{x,z} \qquad z \in \{n_{max} - c, .., n_{max} - 1\}$$

$$\bigwedge \neg enc_{y,t} \qquad t \in \{0, .., c-1\}$$

$$\bigwedge enc_{y,c+s} \rightarrow enc_{x,s} \qquad s \in \{0, .., n_{max} - c - 1\}$$

which can be rewritten as

$$\bigwedge enc_{x,z} \qquad z \in \{n_{max} - c, .., n_{max} - 1\}$$

$$\bigwedge \neg enc_{y,t} \qquad t \in \{0, .., c-1\}$$

$$\bigwedge \neg enc_{y,c+s} \vee enc_{x,s} \qquad s \in \{0, .., n_{max} - c - 1\}$$

The inequalities are therefore transformed in the following formulas:

- Formula 2 becomes

$$\begin{aligned} &lr_{i,j} \rightarrow enc_{x_i,z} && z \in \{W - width_i, .., W - 1\} \\ &lr_{i,j} \rightarrow \neg enc_{x_j,t} && t \in \{0, .., width_i - 1\} \\ &lr_{i,j} \rightarrow (\neg enc_{x_j,width_i+s} \vee enc_{x_i,s}) && s \in \{0, .., W - width_i - 1\} \end{aligned} \tag{6}$$

- Formula 3 becomes

$$\begin{aligned} &lr_{j,i} \rightarrow enc_{x_j,z} && z \in \{W - width_j, .., W - 1\} \\ &lr_{j,i} \rightarrow \neg enc_{x_i,t} && t \in \{0, .., width_j - 1\} \\ &lr_{j,i} \rightarrow (\neg enc_{x_i,width_j+s} \vee enc_{x_j,s}) && s \in \{0, .., W - width_j - 1\} \end{aligned} \tag{7}$$

- Formula 4 becomes

$$\begin{aligned} &du_{i,j} \rightarrow enc_{y_i,z} && z \in \{H - height_i, .., H - 1\} \\ &du_{i,j} \rightarrow \neg enc_{y_j,t} && t \in \{0, .., height_i - 1\} \\ &du_{i,j} \rightarrow (\neg enc_{y_j,height_i+s} \vee enc_{y_i,s}) && s \in \{0, .., H - height_i - 1\} \end{aligned} \tag{8}$$

- Formula 5 becomes

$$\begin{aligned} &du_{j,i} \rightarrow enc_{y_j,z} && z \in \{H - height_j, .., H - 1\} \\ &du_{j,i} \rightarrow \neg enc_{y_i,t} && t \in \{0, .., height_j - 1\} \\ &du_{j,i} \rightarrow (\neg enc_{y_i,height_j+s} \vee enc_{y_j,s}) && s \in \{0, .., H - height_j - 1\} \end{aligned} \tag{9}$$

The first formula of each term can be avoided because the right-hand side of that implication is already made true by the first described constraint, therefore we finally obtain

- Formula 6 becomes

$$\begin{aligned} &\neg lr_{i,j} \vee \neg enc_{x_j,t} \qquad t \in \{0,..,width_i - 1\} \\ &\neg lr_{i,j} \vee \neg enc_{x_j,width_i+s} \vee enc_{x_i,s} \qquad s \in \{0,..,W - width_i - 1\} \end{aligned} \tag{10}$$

- Formula 7 becomes

$$\begin{aligned} &\neg lr_{j,i} \vee \neg enc_{x_i,t} \qquad t \in \{0,..,width_j - 1\} \\ &\neg lr_{j,i} \vee \neg enc_{x_i,width_j+s} \vee enc_{x_j,s} \qquad s \in \{0,..,W - width_j - 1\} \end{aligned} \tag{11}$$

- Formula 8 becomes

$$\begin{aligned} &\neg du_{i,j} \vee \neg enc_{y_j,t} \qquad t \in \{0,..,height_i - 1\} \\ &\neg du_{i,j} \vee \neg enc_{y_j,height_i+s} \vee enc_{y_i,s} \qquad s \in \{0,..,H - height_i - 1\} \end{aligned} \tag{12}$$

- Formula 9 becomes

$$\begin{aligned} &\neg du_{j,i} \vee \neg enc_{y_i,t} \qquad t \in \{0,..,height_j - 1\} \\ &\neg du_{j,i} \vee \neg enc_{y_i,height_j+s} \vee enc_{y_j,s} \qquad s \in \{0,..,H - height_j - 1\} \end{aligned} \tag{13}$$

3.3.1 Implied constraints

Large Rectangles (LR) Implied Constraint If the sum of two rectangles widths is bigger than the width of the strip (i.e. $width_i + width_j > W$) we can assume that they can't be placed one on the left or right side of the other (as explained in Section 1.4.3) and we can delete some of the constraints:
Formula 1 is rewritten as

$$\cancel{lr_{i,j}} \vee \cancel{lr_{j,i}} \vee du_{i,j} \vee du_{j,i}$$

- Formula 10 is deleted

$$\begin{aligned} &\neg lr_{i,j} \vee \neg enc_{x_j,t} \qquad t \in \{0,..,width_i - 1\} \\ &\neg lr_{i,j} \vee \neg enc_{x_j,width_i+s} \vee enc_{x_i,s} \qquad s \in \{0,..,W - width_i - 1\} \end{aligned}$$

- Formula 11 is deleted

$$\begin{aligned} &\neg lr_{j,i} \vee \neg enc_{x_i,t} \qquad t \in \{0,..,width_j - 1\} \\ &\neg lr_{j,i} \vee \neg enc_{x_i,width_j+s} \vee enc_{x_j,s} \qquad s \in \{0,..,W - width_j - 1\} \end{aligned}$$

- Formula 12 is kept

$$\begin{aligned} &\neg du_{i,j} \vee \neg enc_{y_j,t} \qquad t \in \{0, height_i - 1\} \\ &\neg du_{i,j} \vee \neg enc_{y_j,height_i+s} \vee enc_{y_i,s} \qquad s \in \{0, H - height_i - 1\} \end{aligned}$$

- Formula 13 is kept

$$\begin{aligned} &\neg du_{j,i} \vee \neg enc_{y_i,t} \qquad t \in \{0,..,height_j - 1\} \\ &\neg du_{j,i} \vee \neg enc_{y_i,height_j+s} \vee enc_{y_j,s} \qquad s \in \{0,..,H - height_j - 1\} \end{aligned}$$

In this case, we can also apply the same reasoning to the height H(since it is a fixed parameter), because if $height_i + height_j > H$ we can assume that those rectangles can't be placed one above or under the other and we can delete these constraints:
Formula 1 is rewritten as

$$lr_{i,j} \vee lr_{j,i} \vee \cancel{du_{i,j}} \vee \cancel{du_{j,i}}$$

- Formula 10 is kept

$$\neg lr_{i,j} \vee \neg enc_{x_j,t} \qquad t \in \{0,..,width_i - 1\}$$
$$\neg lr_{i,j} \vee \neg enc_{x_j,width_i+s} \vee enc_{x_i,s} \qquad s \in \{0,..,W - width_i - 1\}$$

- Formula 11 is kept

$$\neg lr_{j,i} \vee \neg enc_{x_i,t} \qquad t \in \{0,..,width_j - 1\}$$
$$\neg lr_{j,i} \vee \neg enc_{x_i,width_j+s} \vee enc_{x_j,s} \qquad s \in \{0,..,W - width_j - 1\}$$

- Formula 12 is deleted

$$\neg du_{i,j} \vee \neg enc_{y_j,t} \qquad t \in \{0, height_j - 1\}$$
$$\neg du_{i,j} \vee \neg enc_{y_j,height_i+s} \vee enc_{y_i,s} \qquad s \in \{0, H - height_i - 1\}$$

- Formula 13 is deleted

$$\neg du_{j,i} \vee \neg enc_{y_i,t} \qquad t \in \{0,..,height_j - 1\}$$
$$\neg du_{j,i} \vee \neg enc_{y_i,height_j+s} \vee enc_{y_j,s} \qquad s \in \{0,..,H - height_j - 1\}$$

3.3.2 Symmetry breaking constraints

We've applied 2 kinds of Symmetry Breaking Constraints:

Same Sized Rectangles (SR) Symmetry Breaking If two rectangles are such that $width_i = width_j$ and $height_i = height_j$ we can choose to have rectangle i left-over j (as explained in Section 1.4.3):
Formula 1 is rewritten as

$$lr_{i,j} \vee \cancel{lr_{j,i}} \vee du_{i,j} \vee du_{j,i}$$

- Formula 10 is kept

$$\neg lr_{i,j} \vee \neg enc_{x_j,t} \qquad t \in \{0,..,width_i - 1\}$$
$$\neg lr_{i,j} \vee \neg enc_{x_j,width_i+s} \vee enc_{x_i,s} \qquad s \in \{0,..,W - width_i - 1\}$$

- Formula 11 is deleted

$$\neg lr_{j,i} \vee \neg enc_{x_i,t} \qquad t \in \{0,..,width_j - 1\}$$
$$\neg lr_{j,i} \vee \neg enc_{x_i,width_j+s} \vee enc_{x_j,s} \qquad s \in \{0,..,W - width_j - 1\}$$

- Formula 12 is kept

$$\neg du_{i,j} \vee \neg enc_{y_j,t} \qquad t \in \{0,..,height_i - 1\}$$
$$\neg du_{i,j} \vee \neg enc_{y_j,height_i+s} \vee enc_{y_i,s} \qquad s \in \{0,..,H - height_i - 1\}$$

- Formula 13 is kept

$$\neg du_{j,i} \vee \neg enc_{y_i,t} \qquad t \in \{0,..,height_j - 1\}$$
$$\neg du_{j,i} \vee \neg enc_{y_i,height_j+s} \vee enc_{y_j,s} \qquad s \in \{0,..,H - height_j - 1\}$$

Adding just a new formula:

$$du_{i,j} \rightarrow lr_{j,i}$$

rewritable as

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

Largest Sized Rectangle (LS) Symmetry Breaking We can also assume to have the *largest* rectangle (in terms of area) in the left bottom half of the strip, in order to avoid reflexive symmetries (as explained in Section 1.4.3). This makes it possible to restrict the domain:

$$enc_{x_{largest},i}, \qquad i \in \{\lfloor \frac{W - width_{largest}}{2} \rfloor, W-1\}$$

$$enc_{y_{largest},j}, \qquad j \in \{\lfloor \frac{H - height_{largest}}{2} \rfloor, H-1\}$$

And delete some overlapping constraints for the rectangles i such that $width_i > \lfloor \frac{W - width_{largest}}{2} \rfloor$:
Formula 1 is rewritten as

$$\cancel{lr_{i,largest}} \vee lr_{largest,i} \vee du_{i,largest} \vee du_{largest,i}$$

- Formula 10 is deleted

$$\cancel{\neg lr_{i,largest} \vee \neg enc_{x_{largest},t} \qquad t \in \{0,..,width_i - 1\}}$$
$$\cancel{\neg lr_{i,largest} \vee \neg enc_{x_{largest},width_i+s} \vee enc_{x_i,s} \qquad s \in \{0,..,W - width_i - 1\}}$$

- Formula 11 is kept

$$\neg lr_{largest,i} \vee \neg enc_{x_i,t} \qquad t \in \{0,..,width_largest - 1\}$$
$$\neg lr_{largest,i} \vee \neg enc_{x_i,width_{largest}+s} \vee enc_{x_{largest},s} \qquad s \in \{0,..,W - width_{largest} - 1\}$$

- Formula 12 is kept

$$\neg du_{i,largest} \vee \neg enc_{y_{largest},t} \qquad t \in \{0,..,height_i - 1\}$$
$$\neg du_{i,largest} \vee \neg enc_{y_{largest},height_i+s} \vee enc_{y_i,s} \qquad s \in \{0,..,H - height_i - 1\}$$

- Formula 13 is kept

$$\neg du_{largest,i} \vee \neg enc_{y_i,t} \qquad t \in \{0,..,height_{largest} - 1\}$$
$$\neg du_{largest,i} \vee \neg enc_{y_i,height_{largest}+s} \vee enc_{y_{largest},s} \qquad s \in \{0,..,H - height_{largest} - 1\}$$

and for the rectangles i such that $height_i > \lfloor \frac{H - height_{largest}}{2} \rfloor$:
Formula 1 is rewritten as

$$lr_{i,largest} \vee lr_{largest,i} \vee \cancel{du_{i,largest}} \vee du_{largest,i}$$

- Formula 10 is deleted

$$\neg lr_{i,largest} \vee \neg enc_{x_{largest},t} \qquad t \in \{0,..,width_i - 1\}$$
$$\neg lr_{i,largest} \vee \neg enc_{x_{largest},width_i+s} \vee enc_{x_i,s} \qquad s \in \{0,..,W - width_i - 1\}$$

- Formula 11 is kept

$$\neg lr_{largest,i} \vee \neg enc_{x_i,t} \qquad t \in \{0,..,width_largest - 1\}$$
$$\neg lr_{largest,i} \vee \neg enc_{x_i,width_{largest}+s} \vee enc_{x_{largest},s} \qquad s \in \{0,..,W - width_{largest} - 1\}$$

- Formula 12 is kept

$$\neg du_{i,largest} \vee \neg enc_{y_{largest},t} \qquad t \in \{0,..,height_i - 1\}$$
$$\neg du_{i,largest} \vee \neg enc_{y_{largest},height_i+s} \vee enc_{y_i,s} \qquad s \in \{0,..,H - height_i - 1\}$$

- Formula 13 is deleted

$$\cancel{\neg du_{largest,i} \vee \neg enc_{y_i,t} \qquad t \in \{0,..,height_{largest} - 1\}}$$
$$\cancel{\neg du_{largest,i} \vee \neg enc_{y_i,height_{largest}+s} \vee enc_{y_{largest},s} \qquad s \in \{0,..,H - height_{largest} - 1\}}$$

3.4 Rotation

We've introduced a new variable r for each instance, it is True if the rectangle is rotated, False if it is not. For each of the rectangles, if the height (without rotations) is bigger than the dimension of the whole container, the rotation of that same rectangle is avoided
Therefore, we've rewritten the formulas (0-4) in the following way, considering that if a rectangle is rotated, its height and width are inverted:

- Formula 10 is rewritten as

$$\begin{aligned}
&\neg r_i \rightarrow (\neg lr_{i,j} \vee \neg enc_{x_j,t}) \qquad t \in \{0,..,width_i - 1\} \\
&\neg r_i \rightarrow (\neg lr_{i,j} \vee \neg enc_{x_j,width_i+s} \vee enc_{x_i,s}) \qquad s \in \{0,..,W - width_i - 1\} \\
&r_i \rightarrow (\neg lr_{i,j} \vee \neg enc_{x_j,t}) \qquad t \in \{0,..,height_i - 1\} \\
&r_i \rightarrow (\neg lr_{i,j} \vee \neg enc_{x_j,height_i+s} \vee enc_{x_i,s}) \qquad s \in \{0,..,W - height_i - 1\}
\end{aligned}$$

- Formula 11 is rewritten as

$$\begin{aligned}
&\neg r_j \rightarrow (\neg lr_{j,i} \vee \neg enc_{x_i,t}) \qquad t \in \{0,..,width_j - 1\} \\
&\neg r_j \rightarrow (\neg lr_{j,i} \vee \neg enc_{x_i,width_j+s} \vee enc_{x_j,s}) \qquad s \in \{0,..,W - width_j - 1\} \\
&r_j \rightarrow (\neg lr_{j,i} \vee \neg enc_{x_i,t}) \qquad t \in \{0,..,height_j - 1\} \\
&r_j \rightarrow (\neg lr_{j,i} \vee \neg enc_{x_i,height_j+s} \vee enc_{x_j,s}) \qquad s \in \{0,..,W - height_j - 1\}
\end{aligned}$$

- Formula 12 is rewritten as

$$\begin{aligned}
&\neg r_i \rightarrow (\neg du_{i,j} \vee \neg enc_{y_j,t}) \qquad t \in \{0,..,height_i - 1\} \\
&\neg r_i \rightarrow (\neg du_{i,j} \vee \neg enc_{y_j,height_i+s} \vee enc_{y_i,s}) \qquad s \in \{0,..,H - height_i - 1\} \\
&r_i \rightarrow (\neg du_{i,j} \vee \neg enc_{y_j,t}) \qquad t \in \{0,..,width_i - 1\} \\
&r_i \rightarrow (\neg du_{i,j} \vee \neg enc_{y_j,width_i+s} \vee enc_{y_i,s}) \qquad s \in \{0,..,H - width_i - 1\}
\end{aligned}$$

- Formula 13 is rewritten as

$$\begin{aligned}
&\neg r_j \rightarrow (\neg du_{j,i} \vee \neg enc_{y_i,t}) \qquad t \in \{0,..,height_j - 1\} \\
&\neg r_j \rightarrow (\neg du_{j,i} \vee \neg enc_{y_i,height_j+s} \vee enc_{y_j,s}) \qquad s \in \{0,..,H - height_j - 1\} \\
&r_j \rightarrow (\neg du_{j,i} \vee \neg enc_{y_i,t}) \qquad t \in \{0,..,width_j - 1\} \\
&r_j \rightarrow (\neg du_{j,i} \vee \neg enc_{y_i,width_j+s} \vee enc_{y_j,s}) \qquad s \in \{0,..,H - width_j - 1\}
\end{aligned}$$

3.4.1 Symmetry Breaking

Squares rotation is of course useless (as described in Section 1.4), therefore the variable r_i is removed for each square i.

3.5 Validation

3.5.1 Experimental design

The default setting of Z3 has been used. The device used to benchmark this method is described in Section 1.3.

3.5.2 Experimental results

ins.	Z3	Z3 (SB)	Z3 (rot)	Z3 (rot-SB)
1	**8**	**8**	**8**	**8**
2	**9**	**9**	**9**	**9**
3	**10**	**10**	**10**	**10**
4	**11**	**11**	**11**	**11**
5	**12**	**12**	**12**	**12**
6	**13**	**13**	**13**	**13**
7	**14**	**14**	**14**	**14**
8	**15**	**15**	**15**	**15**
9	**16**	**16**	**16**	**16**
10	**17**	**17**	**17**	**17**
11	**18**	**18**	**18**	**18**
12	**19**	**19**	**19**	**19**
13	**20**	**20**	**20**	**20**
14	**21**	**21**	**21**	**21**
15	**22**	**22**	**22**	**22**
16	**23**	**23**	**23**	**23**
17	**24**	**24**	**24**	**24**
18	**25**	**25**	**25**	**25**
19	**26**	**26**	**26**	27
20	**27**	**27**	**27**	**27**
21	**28**	**28**	**28**	**28**
22	**29**	**29**	30	30
23	**30**	**30**	**30**	**30**
24	**31**	**31**	**31**	**31**
25	**32**	**32**	33	33
26	**33**	**33**	**33**	**33**
27	**34**	**34**	**34**	**34**
28	**35**	**35**	**35**	**35**
29	**36**	**36**	**36**	**36**
30	38	**37**	**37**	38
31	**38**	**38**	**38**	**38**
32	40	40	40	40
33	**40**	**40**	**40**	**40**
34	**40**	**40**	**40**	**40**
35	**40**	**40**	**40**	**40**
36	**40**	**40**	**40**	**40**
37	**60**	**60**	61	61
38	61	**60**	61	61
39	**60**	**60**	**60**	**60**
40	92	91	92	92

Table 2: Table with the best bounds obtained from the SAT solver *Z3*. Results between parentheses and marked with asterisks are instances for which no additional solution has been computed more than the one already computed through the *Bottom-Left-Justified* method to estimate the upper bound, described in section 1.4. The first column corresponds to the base models, while the ones marked with *SB* correspond to the version with Symmetry Breaking techniques, the ones marked with *rot* correspond to the models were rotation is enabled and *rot-SB* marks the ones with rotations and squares symmetry breaking enabled

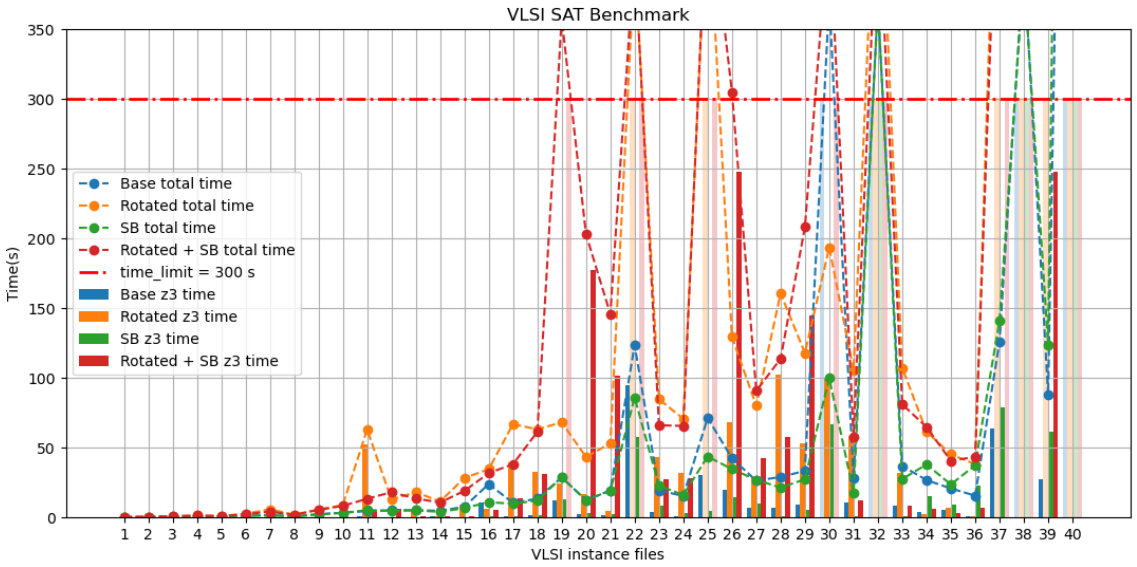

Figure 6: Graph representing the elapsed times during the solving process, using the solver *Z3*. For each instance solved to optimality the bars are colored, while the other bars (not solved to optimality or not solved at all) are grey. The bars correspong to the time taken by the z3 solver alone, while the dotted lines include the time taken by the z3 solver plus the time used to build the sat encoding of the problem through the python script.

The results obtained with the solver *Z3* in the 4 different variants are described by the table 2 (reporting the best value obtained for H for each instance) and by the graphs in Figure 6 (reporting the elapsed time for each instance).

4 SMT Model

4.1 Decision variables

The decision variables correspond to the ones described in Section1.4.

4.2 Objective function

Moreover, We have also defined a variable called H which will be our objective function, representing the height of the rectangle that we want to minimize. H was bounded between an upper and a lower bound similarly to the part in SAT.

4.3 Constraints

First of all the decision variables that reflect the x-coordinate and y-coordinate of each rectangle can only be positive, so we will start with the following constraint:

$$(x_i \geq 0) \wedge (y_i \geq 0) \qquad \forall i \in \{0, .., n_{rectangles} - 1\}$$

no circuit should be placed outside the boundaries of the silicon board, and circuits should not overlap. The first constraint can be stated as follows using first-order logic notation and the width and height of the silicone plate as specified in Section1.4:

$$(x_i + width_i \leq W) \wedge (y_i + height_i \leq H) \qquad \forall i \in \{0, .., n_{rectangles} - 1\}$$

As for the non-overlapping constraints they can be represented as follows:

$$(x_i + width_i \leq x_j) \vee \tag{14}$$
$$(x_j + width_j \leq x_i) \vee \tag{15}$$
$$(y_i + height_i \leq y_j) \vee \tag{16}$$
$$(y_j + height_j \leq y_i) \tag{17}$$
$$\forall i \in \{0, .., n_{rectangles} - 2\}, \forall j \in \{i + 1, .., n_{rectangles} - 1\} \tag{18}$$

The constraints are inspired from the following paper[3]

4.3.1 Implied constraints

Large Rectangles (LR) Implied constraint For rectangles in which the sum of their widths exceed the maximum width (W) of the silicone plate we remove the possibility of packing them in the horizontal direction. The first one will be implemented as follows:

$$\begin{aligned}&(width_i + width_j > W) \Rightarrow ((x_i + width_i > x_j) \wedge (x_j + width_j > x_i)) \\ &\forall i \in \{0, .., n_{rectangles} - 2\}, \forall j \in \{i + 1, .., n_{rectangles} - 1\}\end{aligned} \tag{19}$$

Similarly for the vertical direction we apply the same symmetry breaking. If the sum of the rectangles' heights exceed the upper-bound (H) of the silicon plate's height we implement the symmetry breaking constraint as follows:

$$(height_i + height_j > H) \Rightarrow ((y_i + height_i > x_j) \wedge (y_j + height_j > y_i)) \tag{20}$$
$$\forall i \in \{0, .., n_{rectangles} - 2\}, \forall j \in \{i + 1, .., n_{rectangles} - 1\} \tag{21}$$

4.3.2 Symmetry breaking constraints

Largest Sized Rectangle (LS) Symmetry Breaking In addition, We can flip the plate across the x or y axis and arrive at an equal result given a circuit arrangement. By forcing the circuit with the largest area to be placed in the bottom-left corner of the plate, we can condense the search space. we first find the rectangle with highest area as follows:

$$i_{max} = argmax\{width_i \cdot height_i\} \qquad \forall i \in \{0, .., n_{rectangles} - 1\}$$

Afterwards we ensure it is placed in the bottom left corner of the silicon plate as follows:

$$y_{i_{max}} \leq \frac{H - height_{i_{max}}}{2}$$
$$x_{i_{max}} \leq \frac{W - width_{i_{max}}}{2}$$

However, after studying the results it was observed that symmetry breaking constraints did not lead to solve more instances generally.

4.4 Rotation

The task is unchanged when rectangle rotation is taken into account, however some previously used limitations are changed. At first, we created a R array of Boolean variables. To show whether a rectangle has been rotated, each variable is used. The restrictions employed are the same as those used earlier, but it is now taken into account whether the block is rotated or not in order to solve the problem of rotated rectangles. Following is a redefinition of the constraints: Concerning the non overlapping constraints we added the following in case rotation was true to the non-overlapping constraints and when rotation is not true the constraints remains the same as before:

$$((x_i + height_i \leq x_j) \wedge r_i) \vee \tag{22}$$
$$((x_j + height_j \leq x_i) \wedge r_j) \vee \tag{23}$$
$$((y_i + width_i \leq y_j) \wedge r_i) \vee \tag{24}$$
$$((y_j + width_j \leq y_i) \wedge r_j) \tag{25}$$
$$((x_i + width_i \leq x_j) \wedge \neg r_i) \vee \tag{26}$$
$$((x_j + width_j \leq x_i) \wedge \neg r_j) \vee \tag{27}$$
$$((y_i + height_i \leq y_j) \wedge \neg r_i) \vee \tag{28}$$
$$((y_j + height_j \leq y_i) \wedge \neg r_j) \tag{29}$$
$$\forall i \in \{0, .., n_{rectangles} - 2\}, \forall j \in \{i + 1, .., n_{rectangles} - 1\} \tag{30}$$

Concerning the boundaries of the Silicone Plate the following constraints were added for the x axis:

$$((x_i + width_i \leq W) \wedge \neg r_i) \vee ((x_i + height_i \leq W) \wedge r_i) \qquad \forall i \in \{0, .., n_{rectangles} - 1\}$$

In addition, the following was added to respect the boundaries for the y axsis:

$$((y_i + height_i \leq H) \wedge \neg r_i) \vee ((y_i + height_i \leq H) \wedge r_i) \qquad \forall i \in \{0, .., n_{rectangles} - 1\}$$

We also added an additional symmetry breaking constraint which is not allowing rotation for squares because we will obtain the same result. Implementation is as follows:

$$\begin{aligned} &((width_i == height_i) \Rightarrow \neg r_i) \\ &\forall i \in \{0, .., n_{rectangles} - 1\} \end{aligned} \tag{31}$$

4.5 Validation

4.5.1 Experimental design

Similarly to SAT, Z3 has been used and In Section 1.3 is described the device used to benchmark this method.

4.5.2 Experimental results

ins.	Z3	Z3 (SB)	Z3 (rot)	Z3 (rot-SB)
1	**8**	**8**	**8**	**8**
2	**9**	**9**	**9**	**9**
3	**10**	**10**	**10**	**10**
4	**11**	**11**	**11**	**11**
5	**12**	**12**	**12**	**12**
6	**13**	**13**	**13**	**13**
7	**14**	**14**	**14**	**14**
8	**15**	**15**	**15**	**15**
9	**16**	**16**	**16**	**16**
10	**17**	**17**	**17**	**17**
11	**18**	**18**	-(23**)	-(23**)
12	**19**	**19**	**19**	**19**
13	**20**	**20**	**20**	**20**
14	**21**	**21**	**21**	(29**)
15	**22**	**22**	**22**	**22**
16	**23**	**23**	**23**	(32**)
17	**24**	**24**	**24**	**24**
18	**25**	**25**	**25**	**25**
19	**26**	**26**	-(35**)	-(35**)
20	**27**	**27**	-(38**)	-(38**)
21	**28**	**28**	**28**	-(42**)
22	-(43**)	-(43**)	-(43**)	-(43**)
23	**30**	**30**	-(61**)	-(61**)
24	**31**	**31**	**31**	**31**
25	-(44**)	-(44**)	-(44**)	-(44**)
26	**33**	**33**	-(68**)	**33**
27	**34**	**34**	**34**	**34**
28	**35**	**35**	-(66**)	- **35**
29	**36**	**36**	-(77**)	-(77**)
30	-(78**)	-(78**)	-(78**)	-(78**)
31	**38**	**38**	**38**	-(62**)
32	-(80**)	-(80**)	-(80**)	-(80**)
33	**40**	**40**	**40**	**40**
34	**40**	-(46**)	**40**	-(46**)
35	**40**	**40**	**40**	**40**
36	**40**	**40**	**40**	**40**
37	-(85**)	-(85**)	-(85**)	-(85**)
38	-(88**)	-(88**)	-(88**)	-(88**)
39	-(77**)	-(77**)	-(77**)	-(77**)
40	-(122**)	-(122**)	-(122**)	-(122**)

Table 3: In general 32 instances were solved in the best case. However It is interesting to note that we got better results when we ran each model separately. Specially when rotation was introduced, we were able to solve 27 without symmetry breaking and 25 with. However for benchmarking purposes we had to run all 4 models (base, base-SB, rot, rot-SB) at the same time on the same pc which consumed a lot of computation resources and affected the overall performance.

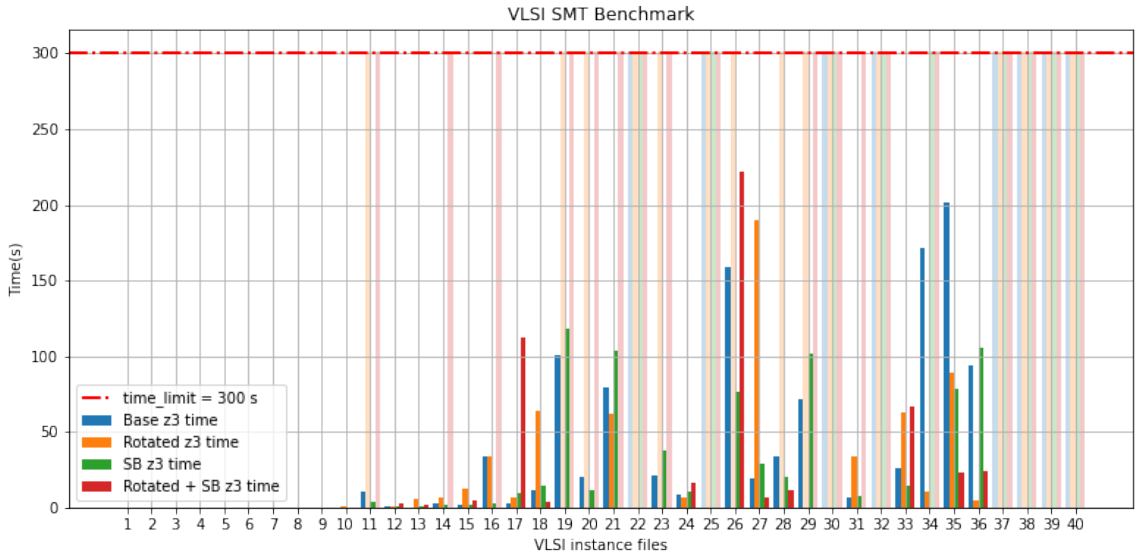

Figure 7: Graph representing the elapsed times during the solving process, using the solver Z3. For each instance solved to optimality the bars are colored, while the other bars (not solved to optimality or not solved at all) are grey.

5 MIP Model

In the project repository and in this file we refer in the same way to this section with *MIP* or *LP*, the reason is that the constraints used are linear and with integer coefficient, therefore this approach belongs to both categories.

5.1 Decision variables

The decision variables correspond to the ones described in Section1.4.

5.2 Objective function

Even in this case, the focus is on the minimization of the function. So, the objective function is just:

$$minimize(H)$$

5.3 Constraints

The model below was implemented as described by *Trespalacios et. al* [4], just the points of reference were changed (in the cited model, the rectangles were placed on a horizontal strip with fixed width and the x and y coordinates were the ones of the left-up corner instead of the left-bottom corner).

$$H_{lb} \leq H \leq H_{ub} \tag{32}$$

$$0 \leq x_i + width_i \leq W \qquad i \in \{0, .., n_{rectangles} - 1\} \tag{33}$$

$$0 \leq y_i + height_i \leq H \qquad i \in \{0, .., n_{rectangles} - 1\} \tag{34}$$

$$x_i + width_i \leq x_j + W(1 - \delta_{i,j,1}) \qquad \forall j > i, \quad i \in \{1..n\}, j \in \{0, .., n_{rectangles} - 1\} \tag{35}$$

$$x_j + width_j \leq x_i + W(1 - \delta_{j,i,1}) \qquad \forall j > i, \quad i \in \{1..n\}, j \in \{0, .., n_{rectangles} - 1\} \tag{36}$$

$$y_i + height_i \leq y_j + H_{ub}(1 - \delta_{i,j,2}) \qquad \forall j > i, \quad i \in \{0, .., n_{rectangles} - 1\}, j \in \{0, .., n_{rectangles} - 1\} \tag{37}$$

$$y_j + height_j \leq y_i + H_{ub}(1 - \delta_{j,i,2}) \qquad \forall j > i, \quad i \in \{0, .., n_{rectangles} - 1\}, j \in \{0, .., n_{rectangles} - 1\} \tag{38}$$

$$\delta_{i,j,1} + \delta_{j,i,1} + \delta_{i,j,2} + \delta_{j,i,2} = 1 \qquad \forall j > i, \quad i \in \{0, .., n_{rectangles} - 1\}, j \in \{0, .., n_{rectangles} - 1\} \tag{39}$$

$$\delta_{i,j,k} \in \{0, 1\}, \qquad \forall i, j \in \{0, .., n_{rectangles} - 1\}, k \in \{0, 1\}$$

Constraint 32 represents the usual bounds of the height, computed as described in Section 1.4.
Constraints 33 and 34 make sure that the rectangles don't go out of the prescribed bounds: W, the fixed width strip and H, the overall strip height to be minimized.
No-overlap Constraints 35, 36, 37, 38, 39 encode the following condition:

$$(x_i + width_i \leq x_j) \bigvee (x_j + width_j \leq x_i) \bigvee (y_i + height_i \leq y_j) \bigvee (y_J + height_j \leq y_i)$$

$$\forall j > i, \quad i \in \{0, .., n_{rectangles} - 1\}, j \in \{0, .., n_{rectangles} - 1\}$$

with the technique highlighted below:

MIP encoding of "or" conditions In MIP, theory doesn't allow *or* ($\bigvee$) conditions, therefore a constraint of the kind

$$(a+b \leq c) \bigvee (c+d \leq a) \bigvee (e+f \leq g) \bigvee (g+h \leq e)$$

is encoded as

$$a+b \leq c+K_1(1-\delta_1)$$
$$c+d \leq a+K_2(1-\delta_2)$$
$$e+f \leq g+K_3(1-\delta_3)$$
$$g+h \leq e+K_4(1-\delta_4)$$

and a constraint that ensures that at least one condition of the above ones is true

$$\delta_1+\delta_2+\delta_3+\delta_4 = 1$$

$$\delta_i \in \{0,1\} \forall i$$

Where K_1, K_2, K_3, K_4 are constants "big enough" to force the satisfiability of the whole condition and δ_i are variables that are 1 if the constraint is satisfied without being forced and 0 if being forced.

With Constraint 39 we require one (and just one) of the conditions to be true (it is the encoding of an *exclusive or*).
From this model we've experimented an alternative formulation of the constraints 35, 36, 37, 38, 39 and we've found one that is performing better:

$$x_i + width_i \leq x_j + W \cdot \delta_{i,j,1} \quad \forall j > i, \quad i \in \{0,..,n_{rectangles}-1\}, j \in \{0,..,n_{rectangles}-1\} \tag{40}$$

$$x_j + width_j \leq x_i + W \cdot \delta_{j,i,1} \quad \forall j > i, \quad i \in \{0,..,n_{rectangles}-1\}, j \in \{0,..,n_{rectangles}-1\} \tag{41}$$

$$y_i + height_i \leq y_j + H_{ub} \cdot \delta_{i,j,2} \quad \forall j > i, \quad i \in \{0,..,n_{rectangles}-1\}, j \in \{0,..,n_{rectangles}-1\} \tag{42}$$

$$y_j + height_j \leq y_i + H_{ub} \cdot \delta_{j,i,2} \quad \forall j > i, \quad i \in \{0,..,n_{rectangles}-1\}, j \in \{0,..,n_{rectangles}-1\} \tag{43}$$

$$\delta_{i,j,1} + \delta_{j,i,1} + \delta_{i,j,2} + \delta_{j,i,2} \leq 3 \quad \forall j > i, \quad i \in \{0,..,n_{rectangles}-1\}, j \in \{0,..,n_{rectangles}-1\} \tag{44}$$

$$\delta_{i,j,1} + \delta_{j,i,1} \geq 1 \quad \forall j > i, \quad i \in \{0,..,n_{rectangles}-1\}, j \in \{0,..,n_{rectangles}-1\} \tag{45}$$

$$\delta_{i,j,2} + \delta_{j,i,2} \geq 1 \quad \forall j > i, \quad i \in \{0,..,n_{rectangles}-1\}, j \in \{0,..,n_{rectangles}-1\} \tag{46}$$

$$\delta_{i,j,k} \in \{0,1\}, \quad \forall i,j \in \{0,..,n_{rectangles}-1\}, k \in \{0,1\}$$

Since $\delta = 1$ for each unsatisfied condition and we know that a rectangle can't be at the same time on the left and on the right of another, we pose the constraint 45. Similarly, we know that a rectangle can't be at the same time under and over another and we pose constraint 46.

5.3.1 Symmetry breaking constraints

Large Rectangles (LR) Symmetry Breaking Same principle highlighted in section 1.4.3. Rectangles i and j such that

$$width_i + width_j > W \qquad i < j, \quad i,j \in \{0,..,n_{rectangles}-1\}$$

are found before the loading of the model and, after having found them, the constraint

$$\delta_{i,j,1} = 1, \quad \delta_{j,i,1} = 1$$

is imposed over those: in this way we avoid to check the situations in which a rectangle is on the left or right side of the other. In this case we don't apply the same constraint on the height H because H is not fixed before starting the solver (as, instead, it was for SAT).

Same Sized Rectangles (SR) Symmetry Breaking Same principle highlighted in Section 1.4.3. If two rectangles are such that $width_i = width_j$ and $height_i = height_j$ we can choose to have rectangle i left-under j: as before, we force this condition through the δ variables.

$$\delta_{j,i,1} = 1, \quad \delta_{j,i,2} = 1$$

Largest Sized Rectangle (LS) Symmetry Breaking Same principle highlighted in Section 1.4.3. We place the *largest* rectangle (in terms of area) in the left bottom half of the strip:

$$x_{largest} \leq \lfloor \frac{W - width_{largest}}{2} \rfloor$$

$$y_{largest} \leq \lfloor \frac{H - height_{largest}}{2} \rfloor$$

Also writable as

$$2x_{largest} \leq W - width_{largest}$$

$$2y_{largest} \leq H - height_{largest}$$

While for rectangles i satisfying the condition $width_i > \lfloor \frac{W - width_{largest}}{2} \rfloor$ we add the following constraint:

$$\delta_{largest,i,1} + \delta_{i,largest,2} + \delta_{largest,i,1} \leq 2$$

Which directly avoids to try configurations in which the rectangle i would be on the left of *largest* when not feasible. In this case we don't apply the same constraint on the height H because H is not fixed before starting the solver (as, instead, it was for SAT).

5.4 Rotations

$$H_{lb} \leq H \leq H_{ub} \tag{47}$$

$$0 \leq x_i + width_{i_{new}} \leq W \qquad i \in \{0, .., n_{rectangles} - 1\} \tag{48}$$

$$0 \leq y_i + height_{i_{new}} \leq H \qquad i \in \{0, .., n_{rectangles} - 1\} \tag{49}$$

$$x_i + width_{i_{new}} \leq x_j + W \cdot \delta_{i,j,1} \qquad \forall j > i, \quad i \in \{0, .., n_{rectangles} - 1\}, j \in \{0, .., n_{rectangles} - 1\} \tag{50}$$

$$x_j + width_{j_{new}} \leq x_i + W \cdot \delta_{j,i,1} \qquad \forall j > i, \quad i \in \{0, .., n_{rectangles} - 1\}, j \in \{0, .., n_{rectangles} - 1\} \tag{51}$$

$$y_i + height_{i_{new}} \leq y_j + H_{ub} \cdot \delta_{i,j,2} \qquad \forall j > i, \quad i \in \{0, .., n_{rectangles} - 1\}, j \in \{0, .., n_{rectangles} - 1\} \tag{52}$$

$$y_j + height_{j_{new}} \leq y_i + H_{ub} \cdot \delta_{j,i,2} \qquad \forall j > i, \quad i \in \{0, .., n_{rectangles} - 1\}, j \in \{0, .., n_{rectangles} - 1\} \tag{53}$$

$$\delta_{i,j,1} + \delta_{j,i,1} + \delta_{i,j,2} + \delta_{j,i,2} \leq 3 \qquad \forall j > i, \quad i \in \{0, .., n_{rectangles} - 1\}, j \in \{0, .., n_{rectangles} - 1\} \tag{54}$$

$$\delta_{i,j,1} + \delta_{j,i,1} \geq 1 \qquad \forall j > i, \quad i \in \{0, .., n_{rectangles} - 1\}, j \in \{0, .., n_{rectangles} - 1\} \tag{55}$$

$$\delta_{i,j,2} + \delta_{j,i,2} \geq 1 \qquad \forall j > i, \quad i \in \{0, .., n_{rectangles} - 1\}, j \in \{0, .., n_{rectangles} - 1\} \tag{56}$$

$$\delta_{i,j,k} \in \{0, 1\}, \qquad \forall i, j \in \{0, .., n_{rectangles} - 1\}, k \in \{0, 1\}$$

$$width_{i_{new}} = r_i \cdot height_i + (1 - r_i) \cdot width_i, \quad i \in \{0, .., n_{rectangles} - 1\} \tag{57}$$

$$height_{i_{new}} = r_i \cdot width_i + (1 - r_i) \cdot height_i, \quad i \in \{0, .., n_{rectangles} - 1\} \tag{58}$$

$$r_i \in \{0, 1\}$$

While the first of part of the model is basically unchanged ($width_i$ and $height_i$ were just substituted by the actual values $width_{i_{new}}$ and $height_{i_{new}}$), the Constraints 57 and 58 are added to swap *width* and *height* in case of rotations, for each rectangle.

5.4.1 Symmetry Breaking

Squares rotation is of course useless (as described in Section 1.4), therefore the following constraint is imposed for each of them:

$$r_i = 0$$

5.5 Validation

5.5.1 Experimental design

We've used the OR-Tools Python API to express, solve the problem and to pass the parameters of the instances and of the $H_U B$ computed.- the default settings of the solvers have been used. The device used to benchmark this method is described in Section 1.3.

5.5.2 Experimental results

ins.	Gurobi	Scip	Gurobi(SB)	Scip(SB)	Gurobi(rot)	Scip(rot)	Gurobi(rot-SB)	Scip(rot-SB)
1	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**
2	**9**	**9**	**9**	**9**	**9**	**9**	**9**	**9**
3	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**
4	**11**	**11**	**11**	**11**	**11**	**11**	**11**	**11**
5	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**
6	**13**	**13**	**13**	**13**	**13**	**13**	**13**	**13**
7	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**
8	**15**	**15**	**15**	**15**	**15**	**15**	**15**	**15**
9	**16**	**16**	**16**	**16**	**16**	**16**	**16**	**16**
10	**17**	**17**	**17**	**17**	**17**	**17**	**17**	**17**
11	**18**	19	**18**	19	**18**	19	**18**	19
12	**19**	**19**	**19**	**19**	**19**	20	**19**	20
13	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**
14	**21**	**21**	**21**	**21**	**21**	**21**	**21**	**21**
15	**22**	**22**	**22**	**22**	**22**	23	**22**	23
16	**23**	24	24	24	**23**	24	**23**	24
17	**24**	25	**24**	25	**24**	25	**24**	25
18	**25**	**25**	**25**	**25**	26	26	26	26
19	**26**	27	**26**	27	27	27	27	27
20	**27**	28	**27**	28	28	28	28	28
21	**28**	30	**28**	29	29	29	29	30
22	**29**	30	30	31	30	30	30	31
23	**30**	31	**30**	31	31	**30**	31	31
24	**31**	32	**31**	**31**	**31**	32	**31**	**31**
25	33	34	33	34	34	34	34	34
26	**33**	34	**33**	34	34	34	34	34
27	**34**	35	**34**	35	**34**	36	**34**	36
28	**35**	36	**35**	**35**	**35**	37	**35**	37
29	**36**	38	**36**	37	37	37	37	39
30	38	41	38	39	38	39	38	40
31	**38**	**38**	**38**	39	**38**	**38**	**38**	39
32	41	42	41	43	40	41	40	42
33	**40**	42	**40**	42	**40**	41	**40**	41
34	41	41	41	41	41	**40**	41	41
35	41	41	**40**	41	**40**	41	**40**	41
36	**40**	41	**40**	41	**40**	41	**40**	41
37	62	62	62	63	61	62	61	62
38	62	63	61	63	61	63	61	62
39	61	63	61	63	61	61	61	63
40	101	- (103**)	101	- (103**)	102	- (103**)	102	- (103**)

Table 4: Table with the best bounds obtained from the MIP solvers *Gurobi* and *Scip*. Results between parentheses and marked with asterisks are instances for which no additional solution has been computed more than the one already computed through the *Bottom-Left-Justified* method to estimate the upper bound, described in section 1.4. The first two columns correspond to the base models, while the ones marked with *SB* correspond to the version with Symmetry Breaking techniques, the ones marked with *rot* correspond to the models were rotation is enabled and *rot-SB* marks the ones with rotations and squares symmetry breaking enabled

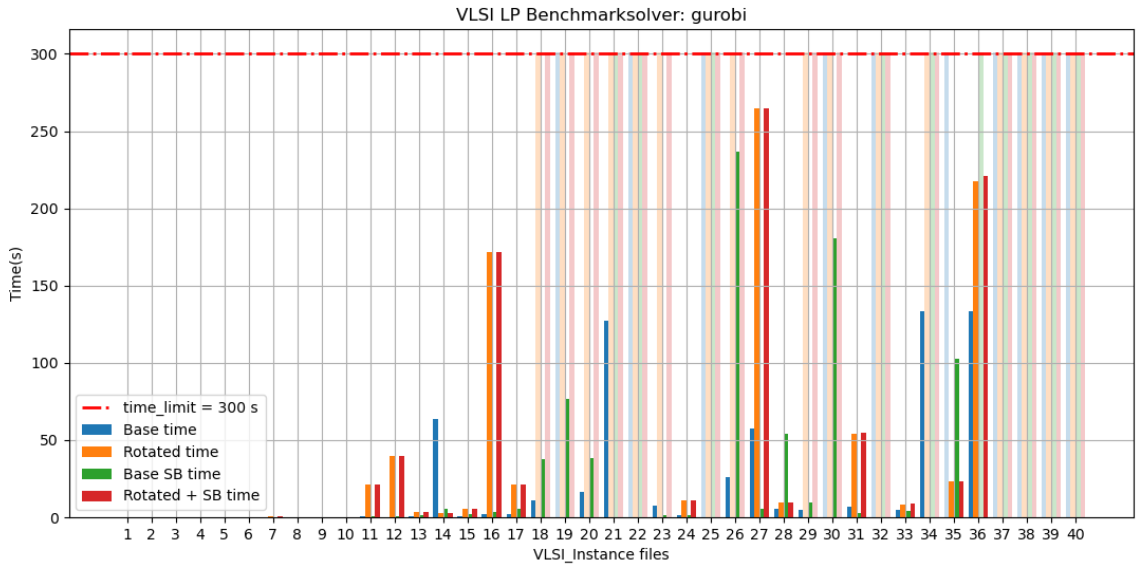

Figure 8: Graph representing the elapsed times during the solving process, using the solver *Gurobi*. For each instance solved to optimality the bars are colored, while the other bars (not solved to optimality or not solved at all) are grey.

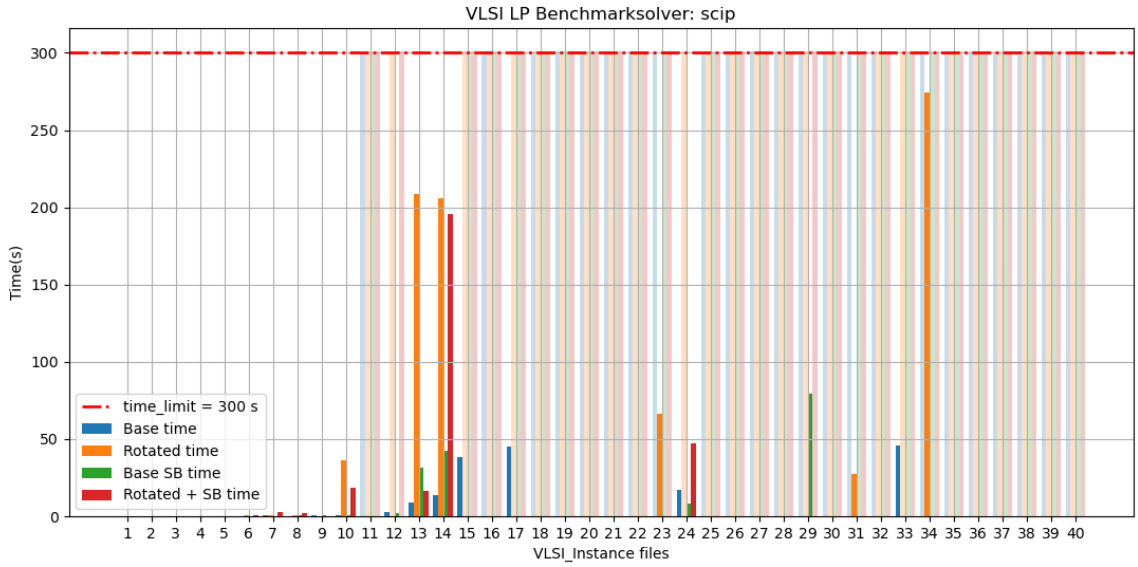

Figure 9: Graph representing the elapsed times during the solving process, using the solver *SCIP*. For each instance solved to optimality the bars are colored, while the other bars (not solved to optimality or not solved at all) are grey.

6 Conclusions

6.1 CP

CP has shown to be the modeling technique with the highest flexibility: this advantage, coupled with the choice of a meaningful solving strategy and the **appropriate global constraints** made it possible to achieve the best results. However, using 'or' constraints would have made the model a bit more intuitive but the solving process much slower: symmetry breaking constraints, in fact, have provided an advantage just in some situations while in others made the solving process slower (because of the application of 'or' and 'implication' constraints). Because of the solving strategy setting, *Chuffed* and *OR-Tools* have reached comparable results, but while *Chuffed* failed to provide solutions when the time limit was reached, *Or-Tools* provided solutions that were even relatively close to the optimal value for H.

6.2 SAT

SAT has also given good results, but it must be taken into account that the conversion of variables and constants from natural to boolean (and viceversa) takes a relevant amount of time[5]. Other than the elapsed time to build and set the model, SAT has required to convert all the constraints in the form of boolean formulas: this format makes it harder for a human to understand, debug and extend the model. In this case, adding constraints that can't be directly expressed as binary formulas may negatively affect the performance (the search space may remarkably increase).

6.3 SMT

introducing symmetry Breaking restrictions to this problem did not always result in improved performance. In many cases, both in the rotated and unrotated case, it actually resulted in the execution timing out while the basic model would have otherwise terminated properly.We can see that, although it is not as effective as the Constraint Programming model, our SMT model can solve 25 of the 40 instances when we allow rotation and 32 of the 40 instances when we don't. For future use, it could be interesting to experiment with a model on these instances that does not include the implied constraints that were mentioned in the project specification because they might be greatly slowing down the computation.

6.4 MIP

In MIP, the commercial solver *Gurobi* has achieved much better results than $SCIP$. In this case the choice of the solver is crucial and we can think of improvements in the performance just through a different way to represent the problem.
Other tentatives, including a benchmark with CPLEX (which are not exhaustively presented in this document, but are in the *other* subfolder of this project) have been made with the $MinizincPythonAPI$ and with the *Gurobipy* API for Gurobi: this API makes it possible to tweak various parameters which are otherwise not directly accessible through OR-Tools or Minizinc, but despite the higher flexibility, no custom setting have achieved better results than the default settings. We can conclude that, other than the choice of the best solver, it is important to have enough expertise to tweak them in the best possible way. Despite its direct encoding of inequality constraints, the encoding of boolean constraints or more complex constraints requires an additional effort (both to encode the model and for the solver).

6.5 General comparison

The degree of complexity of the various instances depends from the number of rectangles to be placed inside the container: the more fragmented that are the rectangular chips (the smaller and the more they're), the harder that is to find the best solution.
All the methods have been successfully applied to solve the prescribed problem[6]. The method that

[5] Python is not efficient to work with boolean numbers and for cycles. The efficiency of the implementation can be improved.

[6] Even if not always providing a solution, but in that case, we can use the positioning used to compute H_{UB} in Section 1.4 as backup solution.

achieved the best results is indeed *CP*: global constraints have been very effective in providing the best solutions even for the most difficult instances. *SAT* is the second best method: this is because to compute the time bound, it must be considered every attempt on the fixed height computed through the bisection method. It is also important to highlight that some solvers such as *Or-Tools* and *Gurobi* can make advantage of the parallelization during the execution, while others, such as *Z3*, are not implemented to support it. Symmetry Breaking has been effective just in some cases: cutting equivalent solutions can sometimes lead to the pruning of solutions which would have been reached before or may lead to an increase of the search space (as in the case of CP). Rotation has in general made the solving process slower: we can say that adding variables increases the search space or the dimensionality of the problem, making it even harder to find a solution. However, this method can be helpful when the rectangular chips have a width $width_i$ which is over the bound W (which, however, is not the case of our dataset instances). Some Machine Learning techniques could be useful to better refine the settings of each solver or to inject a custom heuristic in the CP solver.
The resulting computed heights H are shown below, in Tables 5, 6, 7 and 8, divided by kind of model (*Base*, *Symmetry Breaking*, *Rotations* or *Rotations Symmetry Breaking*), just for the best solvers.

ins.	CP (Or-Tools)	SAT (Z3)	SMT (Z3)	LP (Gurobi)
1	**8**	**8**	**8**	**8**
2	**9**	**9**	**9**	**9**
3	**10**	**10**	**10**	**10**
4	**11**	**11**	**11**	**11**
5	**12**	**12**	**12**	**12**
6	**13**	**13**	**13**	**13**
7	**14**	**14**	**14**	**14**
8	**15**	**15**	**15**	**15**
9	**16**	**16**	**16**	**16**
10	**17**	**17**	**17**	**17**
11	**18**	**18**	**18**	**18**
12	**19**	**19**	**19**	**19**
13	**20**	**20**	**20**	**20**
14	**21**	**21**	**21**	**21**
15	**22**	**22**	**22**	**22**
16	**23**	**23**	**23**	**23**
17	**24**	**24**	**24**	**24**
18	**25**	**25**	**25**	**25**
19	**26**	**26**	**26**	**26**
20	**27**	**27**	**27**	**27**
21	**28**	**28**	**28**	**28**
22	**29**	**29**	-(43**)	**29**
23	**30**	**30**	**30**	**30**
24	**31**	**31**	**31**	**31**
25	**32**	**32**	-(44**)	33
26	**33**	**33**	**33**	**33**
27	**34**	**34**	**34**	**34**
28	**35**	**35**	**35**	**35**
29	**36**	**36**	**36**	**36**
30	**37**	38	-(78**)	38
31	**38**	**38**	**38**	**38**
32	**39**	40	-(80**)	41
33	**40**	**40**	**40**	**40**
34	**40**	**40**	**40**	41
35	**40**	**40**	**40**	41
36	**40**	**40**	**40**	**40**
37	61	**60**	-(85**)	62
38	61	61	-(88**)	62
39	**60**	**60**	-(77**)	61
40	93	92	-(122**)	101

Table 5: Table with the best bounds obtained for all the the methods in the **base** model (no symmetry breaking, no rotations). Results between parentheses and marked with asterisks are instances for which no additional solution has been computed more than the one already computed through the *Bottom-Left-Justified* method to estimate the upper bound, described in section 1.4.

ins.	CP (Or-Tools)(SB)	SAT (Z3)(SB)	SMT (Z3)(SB)	LP (Gurobi)(SB)
1	**8**	**8**	**8**	**8**
2	**9**	**9**	**9**	**9**
3	**10**	**10**	**10**	**10**
4	**11**	**11**	**11**	**11**
5	**12**	**12**	**12**	**12**
6	**13**	**13**	**13**	**13**
7	**14**	**14**	**14**	**14**
8	**15**	**15**	**15**	**15**
9	**16**	**16**	**16**	**16**
10	**17**	**17**	**17**	**17**
11	**18**	**18**	**18**	**18**
12	**19**	**19**	**19**	**19**
13	**20**	**20**	**20**	**20**
14	**21**	**21**	**21**	**21**
15	**22**	**22**	**22**	**22**
16	**23**	**23**	**23**	24
17	**24**	**24**	**24**	**24**
18	**25**	**25**	**25**	**25**
19	**26**	**26**	**26**	**26**
20	**27**	**27**	**27**	**27**
21	**28**	**28**	**28**	**28**
22	**29**	**29**	-(43**)	30
23	**30**	**30**	**30**	**30**
24	**31**	**31**	**31**	**31**
25	**32**	**32**	-(44**)	33
26	**33**	**33**	**33**	**33**
27	**34**	**34**	**34**	**34**
28	**35**	**35**	**35**	**35**
29	**36**	**36**	**36**	**36**
30	38	**37**	-(78**)	38
31	**38**	**38**	**38**	**38**
32	40	40	-(80**)	41
33	**40**	**40**	**40**	**40**
34	**40**	**40**	-(46**)	41
35	**40**	**40**	**40**	**40**
36	**40**	**40**	**40**	**40**
37	61	**60**	-(85**)	62
38	61	**60**	-(88**)	61
39	**60**	**60**	-(77**)	61
40	92	91	-(122**)	101

Table 6: Table with the best bounds obtained for all the the methods in the **Symmetry Breaking** model (with no rotations). Results between parentheses and marked with asterisks are instances for which no additional solution has been computed more than the one already computed through the *Bottom-Left-Justified* method to estimate the upper bound, described in section 1.4.

instance	CP (Or-Tools)(rot)	SAT (Z3)(rot)	SMT (Z3)(rot)	LP (Gurobi)(rot)
1	**8**	**8**	**8**	**8**
2	**9**	**9**	**9**	**9**
3	**10**	**10**	**10**	**10**
4	**11**	**11**	**11**	**11**
5	**12**	**12**	**12**	**12**
6	**13**	**13**	**13**	**13**
7	**14**	**14**	**14**	**14**
8	**15**	**15**	**15**	**15**
9	**16**	**16**	**16**	**16**
10	**17**	**17**	**17**	**17**
11	**18**	**18**	-(23**)	**18**
12	**19**	**19**	**19**	**19**
13	**20**	**20**	**20**	**20**
14	**21**	**21**	**21**	**21**
15	**22**	**22**	**22**	**22**
16	**23**	**23**	**23**	**23**
17	**24**	**24**	**24**	**24**
18	**25**	**25**	**25**	26
19	**26**	**26**	-(35**)	27
20	**27**	**27**	-(38**)	28
21	**28**	**28**	**28**	29
22	**29**	30	-(43**)	30
23	**30**	**30**	-(61**)	31
24	**31**	**31**	**31**	**31**
25	**32**	33	-(44**)	34
26	**33**	**33**	-(68**)	34
27	**34**	**34**	**34**	**34**
28	**35**	**35**	-(66**)	**35**
29	**36**	**36**	-(77**)	37
30	38	**37**	-(78**)	38
31	**38**	**38**	**38**	**38**
32	40	40	-(80**)	40
33	**40**	**40**	**40**	**40**
34	**40**	**40**	**40**	41
35	**40**	**40**	**40**	**40**
36	**40**	**40**	**40**	**40**
37	61	61	-(85**)	61
38	61	61	-(88**)	61
39	61	**60**	-(77**)	61
40	98	92	-(122**)	102

Table 7: Table with the best bounds obtained for all the the methods in the **Rotations** model (with no Symmetry Breaking). Results between parentheses and marked with asterisks are instances for which no additional solution has been computed more than the one already computed through the *Bottom-Left-Justified* method to estimate the upper bound, described in section 1.4.

ins.	CP (Or-Tools)(rot-SB)	SAT (Z3)(rot-SB)	SMT (Z3)(rot-SB)	LP (Gurobi)(rot-SB)
1	**8**	**8**	**8**	**8**
2	**9**	**9**	**9**	**9**
3	**10**	**10**	**10**	**10**
4	**11**	**11**	**11**	**11**
5	**12**	**12**	**12**	**12**
6	**13**	**13**	**12**	**13**
7	**14**	**14**	**14**	**14**
8	**15**	**15**	**15**	**15**
9	**16**	**16**	**16**	**16**
10	**17**	**17**	**17**	**17**
11	**18**	**18**	-(23**)	**18**
12	**19**	**19**	**19**	**19**
13	**20**	**20**	**20**	**20**
14	**21**	**21**	-(29**)	**21**
15	**22**	**22**	**22**	**22**
16	**23**	**23**	-(32**)	**23**
17	**24**	**24**	**24**	**24**
18	**25**	**25**	**25**	26
19	**26**	**26**	-(35**)	27
20	**27**	**27**	-(38**)	28
21	**28**	**28**	-(42**)	29
22	**29**	30	-(43**)	30
23	**30**	**30**	-(61**)	31
24	**31**	**31**	**31**	**31**
25	**32**	33	-(44**)	34
26	**33**	**33**	**33**	34
27	**34**	**34**	**34**	**34**
28	**35**	**35**	**35**	**35**
29	**36**	**36**	-(77**)	37
30	38	**37**	-(78**)	38
31	**38**	**38**	-(62**)	**38**
32	40	40	-(80**)	40
33	**40**	**40**	**40**	**40**
34	**40**	**40**	-(46**)	41
35	**40**	**40**	**40**	**40**
36	**40**	**40**	**40**	**40**
37	61	61	-(85**)	61
38	61	61	-(88**)	61
39	61	**60**	-(77**)	61
40	98	92	-(122**)	102

Table 8: Table with the best bounds obtained for all the the methods in the **Rotations Symmetry Breaking** model. Results between parentheses and marked with asterisks are instances for which no additional solution has been computed more than the one already computed through the *Bottom-Left-Justified* method to estimate the upper bound, described in section 1.4.

References

[1] Brenda S. Baker, E. G. Coffman, Jr., and Ronald L. Rivest. Orthogonal packings in two dimensions. *SIAM Journal on Computing*, 9(4):846–855, 1980.

[2] Takehide Soh, Katsumi Inoue, Naoyuki Tamura, Mutsunori Banbara, and Hidetomo Nabeshima. A sat-based method for solving the two-dimensional strip packing problem. *Fundam. Inform.*, 102:467–487, 01 2010.

[3] Suchandra Banerjee, Anand Ratna, and Suchismita Roy. Satisfiability modulo theory based methodology for floorplanning in vlsi circuits, 2017.

[4] Francisco Trespalacios and Ignacio E. Grossmann. Symmetry breaking for generalized disjunctive programming formulation of the strip packing problem. *Annals of Operations Research*, 258(2):747–759, 11 2017.