1 Introduction

Very large-scale integration (VLSI) is the process of creating an integrated circuit by combining millions or billions of transistors onto a single chip. A typical example is the smartphone. The modern trend of shrinking transistors sizes, allowing engineers to fit more and more transistors into the same area of silicon, has pushed the integration of more and more functions of cellphone circuitry into a single silicon plate. This enables the modern cellphone to mature into a powerful tool that shrank from the size of a large brick-sized unit to a device small enough to comfortably carry in a pocket or purse, with a video camera, touchscreen and other advanced features.
We will describe in this report our approach to design the VLSI of the circuits defining their electrical device: given a fixed-width plate and a list of rectangular circuits, decide how to place them on the plate so that the length of the final device is minimized, therefore improving its portability.
We will consider two variants of the problem:

- Each circuit must be placed in a fixed orientation with respect to the others. This means that an $n \times m$ circuit cannot be positioned as an $m \times n$ circuit in the silicon plate.
- In the second case, the rotation is allowed, which means that an $n \times m$ circuit can be positioned either as it is or as $m \times n$.

In order to tackle this optimization problem we used Constraint Programming (CP), propositional SATisfiability (SAT), its extension to Satisfiability Modulo Theories (SMT) and in the end Mixed-Integer Linear Programming (MIP). The reader can find more about the mentioned techniques and approaches in the specific sections.

Each different approach will take as input different VLSI instances. That is, a text file consisting of lines of integer values. The first line gives the width of the silicon plate. The following line give the number of circuits to place inside the plate. The following lines represent respectively the horizontal and vertical dimensions of the *i*-th circuit. For example, a file with the lines:

```
9
5
3 3
2 4
2 8
3 9
4 12
```

describes an instance in which the silicon plate has the width 9 and we need to install 5 circuits, with dimensions 3 3, 2 4, 2 8, 3 9, 4 12.
Moreover, each model will offer as output a solution following a given format. In fact, where to place a circuit *i* can be determined by the position of the circuit in the silicon plate. The solution indicates also the length of the plate, as well as the positions of *i* given its bottom-left coordinates. For example we could have the following solution:

9 12
5
3 3 4 0
2 4 7 0
2 8 7 4
3 9 4 3
4 12 0 0

which says that the left-bottom corner of the 3 × 3 circuit is positioned at (4,0). The solution can be represented as we can see in Figure 13.

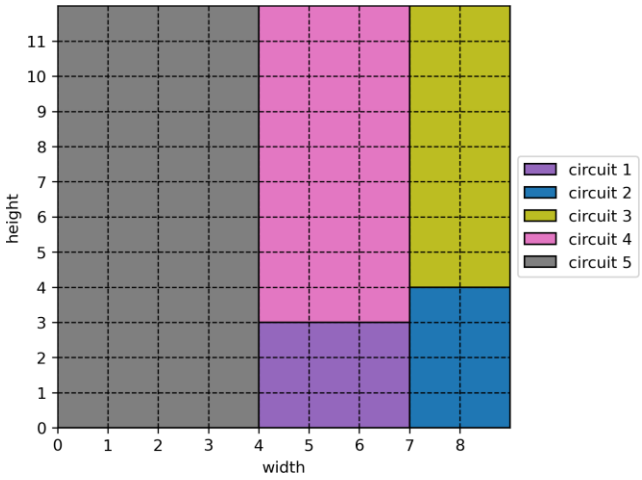


Figure 1: Graphical example of the instance solution given above.

2 Constraint Programming

2.1 Decision variables

Constraint Programming is a paradigm for solving combinatorial problems. In order to solve the VLSI problem using this technique, we need to declaratively state the constraints on the feasible solutions for a set of decision variables, besides a method to solve these constraints.
Therefore, the following constant variables have been defined:

- x: width of each circuit.
- y: height of each circuit.
- w: width of the silicon chip.
- n: number of circuits to be placed inside the plate.

Moreover, the following necessary decision variables have been taken into account:

- x_coord: the x-coordinate of a given circuit on the plate.
- y_coord: the y-coordinate of a given circuit on the plate.

The two above defined decision variables fall in the following domains:

$$x\_coord \in [0, max\_width\_limit - min(x)]$$

$$y\_coord \in [0, max\_height\_limit - min(y)]$$

Where:

$$max\_width\_limit = min(w, \sum_{i=0}^{n} x_i)$$

and max_height_limit is defined in 2.

2.2 Objective function

The objective variable we want to minimize is *sol*, that is the total height of the plate. Our objective function is to minimize the height of the chip:

$$min\_height\_limit \leq sol \leq max\_height\_limit$$

Where:

$$min\_height\_limit = \frac{\sum_{i=0} x_i * y_i}{max\_width\_limit} \quad (1)$$

$$max\_height\_limit = \sum_{i=0} y_i \quad (2)$$

min_height_limit represents the case in which the circuits are placed in such a way that there are no blank spaces in the silicon chip. This minimum limit is useful as the solver won't look for configurations that are unfeasible. On the other side, *max_height_limit* represents the case in which all the circuits are on top of each other.

2.3 Constraints

All circuits locations must be positive:

$$x\_coord_i \geq 0, \qquad \forall\ i, \text{ with } i \in [1, n]$$

$$y\_coord_i \geq 0, \qquad \forall\ i, \text{ with } i \in [1, n]$$

All locations must be inside the silicon chip:

$$x\_coord_i + x_i \leq w, \qquad \forall\ i, \text{ with } i \in [1, n]$$

$$y\_coord_i + y_i \leq sol, \qquad \forall\ i, \text{ with } i \in [1, n]$$

No two circuits must overlap:

$$diffn(x\_coord_i, y\_coord_i, x_i, y_i) \qquad \forall\ i, \text{ with } i \in [1, n]$$

Cumulative constraints are used for the x and y-axis. Those kind of constraints ensures that the tasks taken into account are placed in such a way that the amount of occupied resources never exceeds the given limit.
In this case, for the y-axis the starting time is the y-coordinate of the bottom-left corner of each circuit, the duration is how much it extends in the y-axis (that is, the height) and the resources occupied is the extension on the x-axis (width), which, cumulative, must not be over the upper bound of the width. For the x-axis, the x-coordinate, width, height, and total height are used to enforce the following constraint:

$$cumulative(y\_coord_i, y_i, x_i, w) \qquad \forall\ i, \text{ with } i \in [1, n]$$

$$constraint\_cumulative(x\_coord_i, x_i, y_i, w) \qquad \forall\ i, \text{ with } i \in [1, n]$$

2.3.1 Symmetry Breaking Constraints

As the chip is expected to be rectangular or squared, if we geometrically reflect a solution on its *x* or *y* median axis, we get another valid solution. This unwanted symmetry is handled using the *lex_less*

symmetry breaking constraints as follows, even though there is not improvement in the total running time to fine the solution:

$$lex\_less(x\_coord_i, (w - x_i)) \qquad \forall\ i, \text{ with } i \in [1, n]$$

$$lex\_less(y\_coord_i, (max\_height\_limit - x_i)) \qquad \forall\ i, \text{ with } i \in [1, n]$$

2.4 Rotation

In order to consider also the rotation, we take into account the possibility to swap x_i with y_i of each circuit i. For this, we can consider two arrays (x_dim, y_dim) and (y_dim, x_dim), so that our model can take any pair (x, y) from the above pairs of arrays. For that, we need to re-define the variables, their domains and add a few more constraints as follows:

- rotated: an array composed of booleans, in which $rotated_i = True$ means that the circuit i is rotated.
- x_dim, y_dim: two arrays of integers variables that contain at index i respectively rotated width and rotated height of circuit i. The rotated dimensions are computed simply with:

$$x\_dim = x\_original * (1 - rotated_i) + y\_original * rotated_i$$
$$y\_dim = y\_original * (1 - rotated_i) + x\_original * rotated_i$$

- The maximum width limit is defined as $limit\_dim$:

$$limit\_dim = min(w, sum(max(x\_original_i, y\_original_i)))$$

  Where $x\_original_i$ and $y\_original_i$ are the original coordinates of the circuit i not rotated.

- Maximum height limit is:

$$limit\_h = sum(max(x\_original_i, y\_original_i))$$

- Minimum height is:

$$min\_height\_limit = \frac{\sum_{i=1}^{n} x_i * y_i}{max\_width\_limit}$$

- For each i-th circuit, the domain of x_coord becomes the following:

$$0 \leq x\_coord_i \leq limit\_dim - min(x\_original + +y\_original)$$

- Similar reasoning for y_coord:

$$0 \leq y\_coord_i \leq limit\_h - min(x\_original + +y\_original)$$

- The rest of the code is mostly the same as the one without rotation. In symmetry breaking, we added an addition constraint indicating that if the circuit is a square then it should not be rotated.

2.5 Validation

The model was run using Gecode. The experimental design and their respective results are described in the following sections.

2.5.1 Experimental design

We have used *int_search*(*variables, variable choice, constraint choice*) for our experiments. However, for this project we have experimented with different combinations of variables and value choices using both Gecode and Chuffed.
The best results were obtained with Gecode, used in parallel mode and with four processes. Also with *dom_w_deg* as variable choice and *indomain_random* as value choice selections.

2.5.2 Experimental results

We got the following results:

ID	Gecode + SB (1 T)	Gecode + SB (4 T)	Sol	Gecode w/o SB (1 T)	Gecode w/o SB (4 T)	Sol
1	**0.4**	**0.4**	**8**	**0.4**	**0.4**	**8**
2	**0.3**	**0.4**	**9**	**0.4**	**0.3**	**9**
3	**0.3**	**0.4**	**10**	**0.4**	**0.3**	**10**
4	**0.3**	**0.4**	**11**	**0.5**	**0.3**	**11**
5	**0.3**	**0.5**	**12**	**0.5**	**0.3**	**11**
6	**0.4**	**0.4**	**13**	**0.5**	**0.3**	**13**
7	**0.3**	**0.4**	**14**	**0.6**	**0.3**	**14**
8	**0.3**	**0.5**	**15**	**0.5**	**0.4**	**15**
9	**0.3**	**0.4**	**16**	**0.6**	**0.5**	**16**
10	**0.5**	**0.8**	**17**	**0.9**	**0.5**	**17**
11	**108.1**	**249**	**18**	**9.9**	**103.8**	**18**
12	**0.8**	**0.6**	**19**	**0.5**	**0.5**	**19**
13	**11.8**	**12**	**20**	**4.4**	**7.9**	**20**
14	**1.0**	**1.1**	**21**	**0.7**	**0.6**	**21**
15	**0.6**	**1.0**	**22**	**10.3**	**2.1**	**22**
16	Timeout - satisfied	Timeout - satisfied	N/A	Timeout - satisfied	Timeout - satisfied	N/A
17	Timeout - satisfied	**274.1**	**24**	**1.3**	**0.8**	**24**
18	Timeout - satisfied	**179.3**	**25**	Timeout - satisfied	Timeout - satisfied	N/A
19	**49.9**	**47.5**	**26**	**1.7**	**1.5**	**26**
20	Timeout - satisfied	Timeout - satisfied	N/A	Timeout - satisfied	**256.5**	**27**
21	Timeout - satisfied	Timeout - satisfied	N/A	**43.6**	**199.0**	**28**
22	Timeout - satisfied	Timeout - satisfied	N/A	**81**	**22.4**	**29**
23	Timeout - satisfied	**197.8**	**30**	**9.9**	**10.2**	**30**
24	**37.6**	**21.4**	**31**	**0.5**	**0.8**	**31**
25	Timeout - satisfied	Timeout - satisfied	N/A	Timeout - satisfied	Timeout - satisfied	N/A
26	Timeout - satisfied	Timeout - satisfied	N/A	**81.5**	**45.7**	**33**
27	**39.2**	**50.6**	**34**	**0.8**	**1.0**	**34**
28	Timeout - satisfied	Timeout - satisfied	N/A	**0.8**	**1.0**	**35**
29	Timeout - satisfied	Timeout - satisfied	N/A	**37.9**	**4.6**	**36**
30	Timeout - satisfied	Timeout - satisfied	N/A	Timeout - satisfied	Timeout - satisfied	N/A
31	Timeout - satisfied	Timeout - satisfied	N/A	**5.9**	**1.4**	**38**
32	Timeout - satisfied	Timeout - satisfied	N/A	Timeout - satisfied	Timeout - satisfied	N/A
33	Timeout - satisfied	Timeout - satisfied	N/A	**2.0**	**2.2**	**40**
34	Timeout - satisfied	Timeout - satisfied	N/A	Timeout - satisfied	Timeout - satisfied	N/A
35	Timeout - satisfied	Timeout - satisfied	N/A	Timeout - satisfied	Timeout - satisfied	N/A
36	Timeout - satisfied	Timeout - satisfied	N/A	Timeout - satisfied	Timeout - satisfied	N/A
37	Timeout - satisfied	Timeout - satisfied	N/A	Timeout - satisfied	Timeout - satisfied	N/A
38	Timeout - satisfied	Timeout - satisfied	N/A	Timeout - satisfied	Timeout - satisfied	N/A
39	Timeout - satisfied	Timeout - satisfied	N/A	Timeout - satisfied	Timeout - satisfied	N/A
40	Timeout - satisfied	Timeout - satisfied	N/A	Timeout - satisfied	Timeout - satisfied	N/A

Table 2: Results obtained with the standard model, without rotation.

In the following Table, instead, we can see the results obtained with the rotation model.

ID	Gecode + SB (1 T)	Gecode + SB (4 T)	Sol	Gecode w/o SB (1 T)	Gecode w/o SB (4 T)	Sol
1	**0.3**	**0.3**	**8**	**0.4**	**0.3**	**8**
2	**0.3**	**0.4**	**9**	**0.3**	**0.4**	**9**
3	**0.4**	**0.3**	**10**	**0.3**	**0.3**	**10**
4	**0.3**	**0.3**	**11**	**0.3**	**0.3**	**11**
5	**0.3**	**0.3**	**12**	**0.3**	**0.4**	**12**
6	**0.4**	**0.8**	**13**	**0.6**	**0.3**	**13**
7	**7.8**	**3.2**	**14**	**9.2**	**242.5**	**14**
8	**0.4**	**0.4**	**15**	**0.5**	**0.4**	**15**
9	**1.3**	**1.9**	**16**	**1.9**	**1.1**	**16**
10	Timeout - satisfied	**0.5**	**17**	Timeout - satisfied	Timeout - satisfied	N/A
11	Timeout - satisfied	Timeout - satisfied	N/A	Timeout - satisfied	**65.8**	**18**
12	Timeout - satisfied	Timeout - satisfied	N/A	Timeout - satisfied	**89.7**	**19**
13	Timeout - satisfied	Timeout - satisfied	N/A	Timeout - satisfied	**106.8**	**20**
14	**97.5**	**118.3**	**21**	Timeout - satisfied	**164.3**	**21**
15	Timeout - satisfied	Timeout - satisfid	N/A	Timeout - satisfied	**11.0**	**22**
16	Timeout - satisfied	Timeout - satisfied	N/A	Timeout - satisfied	Timeout - satisfied	N/A
17	Timeout - satisfied	Timeout - satisfied	N/A	Timeout - satisfied	Timeout - satisfied	N/A
18	Timeout - satisfied	Timeout - satisfied	N/A	Timeout - satisfied	Timeout - satisfied	N/A
19	Timeout - satisfied	Timeout - satisfied	N/A	Timeout - satisfied	Timeout - satisfied	N/A
20	Timeout - satisfied	Timeout - satisfied	N/A	Timeout - satisfied	Timeout - satisfied	N/A
21	Timeout - satisfied	Timeout - satisfied	N/A	Timeout - satisfied	Timeout - satisfied	N/A
22	Timeout - satisfied	Timeout - satisfied	N/A	Timeout - satisfied	Timeout - satisfied	N/A
23	Timeout - satisfied	Timeout - satisfied	N/A	Timeout - satisfied	Timeout - satisfied	N/A
24	Timeout - satisfied	Timeout - satisfied	N/A	Timeout - satisfied	**6.0**	**31**
25	Timeout - satisfied	Timeout - satisfied	N/A	Timeout - satisfied	Timeout - satisfied	N/A
26	Timeout - satisfied	Timeout - satisfied	N/A	Timeout - satisfied	Timeout - satisfied	N/A
27	**28.4**	Timeout - satisfied	**34**	Timeout - satisfied	**28.6**	**34**
28	Timeout - satisfied	Timeout - satisfied	N/A	Timeout - satisfied	Timeout - satisfied	N/A
29	**159.5**	Timeout - satisfied	**36**	**2.3**	Timeout - satisfied	**36**
30	Timeout - satisfied	Timeout - satisfied	N/A	Timeout - satisfied	Timeout - satisfied	N/A
31	Timeout - satisfied	Timeout - satisfied	N/A	Timeout - satisfied	Timeout - satisfied	N/A
32	Timeout - satisfied	Timeout - satisfied	N/A	Timeout - satisfied	Timeout - satisfied	N/A
33	**245.6**	Timeout - satisfied	**40**	Timeout - satisfied	Timeout - satisfied	N/A
34	Timeout - satisfied	Timeout - satisfied	N/A	Timeout - satisfied	Timeout - satisfied	N/A
35	Timeout - satisfied	Timeout - satisfied	N/A	Timeout - satisfied	Timeout - satisfied	N/A
36	Timeout - satisfied	Timeout - satisfied	N/A	Timeout - satisfied	**99.4**	**40**
37	Timeout - satisfied	Timeout - satisfied	N/A	Timeout - satisfied	Timeout - satisfied	N/A
38	Timeout - satisfied	Timeout - satisfied	N/A	Timeout - satisfied	Timeout - satisfied	N/A
39	Timeout - satisfied	Timeout - satisfied	N/A	Timeout - satisfied	Timeout - satisfied	N/A
40	Timeout - satisfied	Timeout - satisfied	N/A	Timeout - satisfied	Timeout - satisfied	N/A

Table 3: Results obtained with the rotation model.

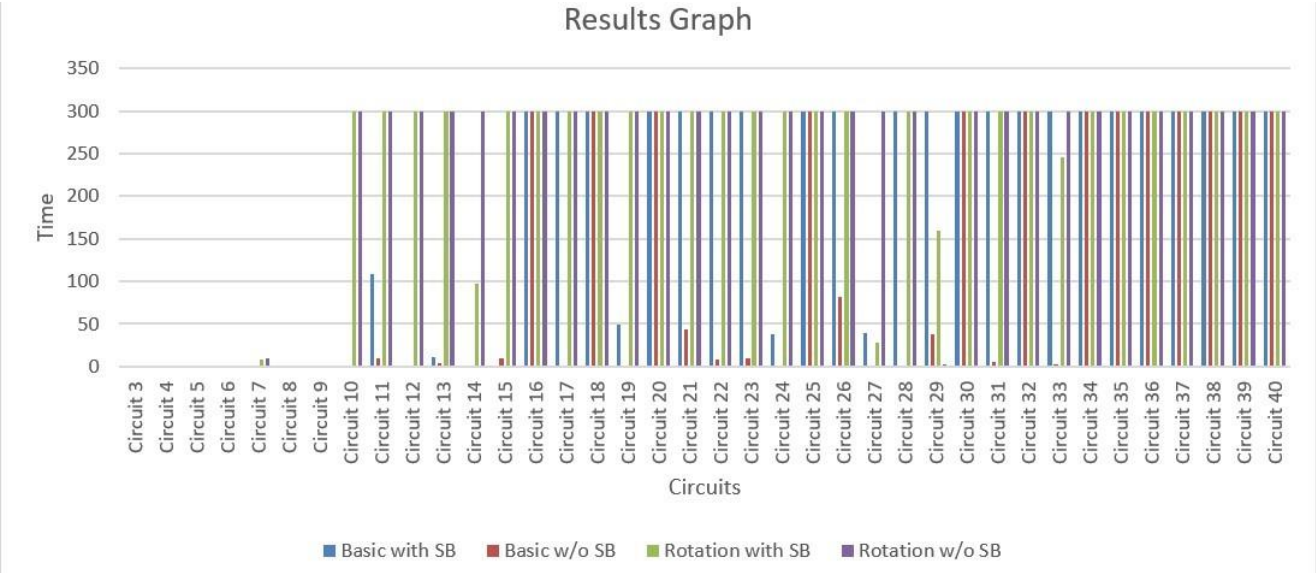


Figure 2: Graphical results of the running time obtained with the CP model.

3 SAT Model

The Boolean Satisfiability problem is a kind of math-based problem in which it is determined whether there is an interpretation that satisfies a Boolean Formula. A formula is satisfiable if all of its variables can be replaced by True or False in such a way that the whole formula is True. However, if no assignment of variables exists that makes the formula True, it is called unsatisfiable. SAT is one of the most discussed problems in logic and computer science and it was the first problem to be proved NP-complete. Solving a wide range of optimization problems are as difficult as SAT since these problems can be reduced to a SAT problem. Heuristic SAT algorithms are able to solve complex problems such as circuit design and artificial intelligence problems that may contain tens of thousands of variables. In this section as another approach, SAT is used to address the VLSI problem, where the model consists of a Python file in which boolean variables and constraints on the defined variables are expressed as classes of the z3 library.

3.1 Decision variables

In SAT, the modelling process is done using formulae in propositional logic composed of boolean variables and the logical operators that in this case are expressed using relational operators provided by Z3Py library.
The definition of the variables is somehow driven by the project specifications:

- **r**: rectangle/ circuits to place,
- **width/height**: the actual width/height of each rectangle,
- **W**: the maximum possible width of the silicon chip.
- **H**: maximum height is, for every instance, computed the summation of the area of all rectangles divided by the width of the silicon plate, all rounded to the ceiling value:

$$H = \frac{\sum_{i=1}^{n\_rectangles} width_i * height_i}{W}$$

3.2 Objective function

The definition of the variables representing the bottom-left corner of the rectangles is not straightforward like the array defined in CP. Indeed, SAT works only on boolean variables. Hence, we could start from the CP model and, through Order encoding [1], we get the SAT model based on the mentioned reference. Now we will briefly explain how order encoding was implied in our SAT model:

$$(x \leq 0) \vee \neg(y \leq 1).$$

Those constraints are translated into formula of a SAT problem:

$$PosX_{i,0} \wedge PosY_{i,1}$$

Meaning: $PosX_{i,0}$ is true if $X_i \leq 0$. Same stands for $PosY_{i,1}$, that is true if $Y_i \leq 1$ and so on.
Order encoding fits well in this project, because it interprets in a more natural way the order relation of integers which is useful for non-overlapping constraints. In our model we have defined two 2D-matrices of boolean variables, *PosX* and *PosY* , with dimensions respectively ($W \times n\_rectangles$) and ($H \times n\_rectangles$), representing the encoded coordinates *x* and *y* of the bottom-left corner of each rectangle/circuit.
Those matrices are not the only variables needed for solving the SAT problem; we need, in addition, two other matrices, with the same dimension ($n\_blocks \times n\_blocks$), represented by:

- $left_{i,j}$: it is True if given two rectangles r_i and r_j with $i \neq j$, r_i is placed to the left of r_j.
- $upper_{i,j}$: it returns True if the board r_j is placed above r_i.

3.3 Constraints

Our model is constructed based on the 2D perspective. Using the variables that have been defined it was possible to impose the main constraints of the problem.

3.3.1 The order encoding constraints

The following 2-literal axiom clauses are assigned for each rectangle r_i, with:

$$\neg PosX_{i,x} \vee PosX_{i,x+1} \quad \neg PosY_{i,y} \vee PosX_{i,y+1}$$

Where:

- W is the maximum width and H is the maximum height as defined before.
- $0 \leq x < W - width_{r_i}$
- $0 \leq y < H - height_{r_i}$

3.3.2 Non-overlapping constraints

To avoid the possible overlapping of circuits; the main idea was to impose that each cell in the plate has at most one true value. In this way circuits cannot be placed in the same position. So based on the Order-Encoding, the following constraint are defined:
By 4-literal clause:

$$left_{i,j} \vee left_{j,i} \vee upper_{i,j} \vee upper_{j,i}$$

By 3-literal clause:

$$\neg left_{i,j} \vee PosX_{i,x} \vee \neg PosX_{j,x+width_i}$$
$$\neg left_{j,i} \vee PosX_{j,x} \vee \neg PosX_{i,x+width_j}$$
$$\neg upper_{i,j} \vee PosY_{i,y} \vee \neg PosY_{j,y+height_i}$$
$$\neg upper_{j,i} \vee PosY_{j,y} \vee \neg PosY_{i,y+height_j}$$

3.3.3 Additional constraints

Given two rectangles, if the sum of the widths is greater than W, then the two rectangles cannot stay side by side:

$$\neg left_{i,j} \wedge \neg left_{j,i} \text{ if } width_i + width_j > W$$

In case the sum of the heights is greater than W, then the two rectangles cannot stay also one above the other:

$$\neg upper_{i,j} \wedge \neg upper_{j,i} \text{ if } height_i + height_j > H$$

3.3.4 Symmetry Breaking Constraints

In order to break the symmetries of two interchangeable rectangles whose have the same width and height, we encode the following constraints:

$$\neg left_{j,i}$$
$$left_{i,j} \vee \neg upper_{j,i}$$

Those means that given two rectangles r_i and r_j with $i \neq j$, with same height and width, we impose that r_j must not be on the left of r_i and either r_i is on the left, or r_i must be not under the rectangle r_j.

3.4 Rotation model

We also need to handle the case when the circuits are rotated inside the plate, which means that a $w \times h$ circuit can be positioned as $h \times w$. To do that we had to define another model with some minor changes during the Encoding.
In this modified approach we have two matrices *RotatedX* and *RotatedY* , with dimensions respectively $W \times n$ *rectangles* and $H \times n$ *rectangles*, which are the modification of the two predefined matrices *PosX PosY* , in a way that *RotatedX* is actually the Y-coordinates instead of *X* of the left corner of each circuit, as same for *RotatedY* which becomes the representation of the X-coordinates instead of Y. And by that we have rotated the rectangle r_i by 90°. The same constraints as also same symmetry breaking applied for the standard model hold also for the rotation one.

3.5 Validation

The experimental designs and their respective results are described in the following sections.

3.5.1 Experimental design

We used the Z3 Library in Python in order to solve the problem. Two modules were implemented with and without Rotation, but before there we make sure to sort the circuits inside each plate sorting the area which would facilitate the process of computing the solution by using, or not using, the symmetry breaking constraints.

3.5.2 Experimental results

In these table it is shown how much does the solver takes to compute the solution by using, or not using, the symmetry breaking constraints in order to achieve the optimality wich is actually achieved by the lowest plate height between the implemented models. In the Table 4 we can see the results obtained with this approach. The legend is the following:

- N/A: No answer is obtained within the time limit.
- H: The optimal plate height achieved.
- SB: Model with Symmetry Breaking constraints.
- SAT w/o SB: Model without Symmetry Breaking constraints.
- Value in bold: Emphasizes that the instance is solved to optimality.

	SAT + SB		SAT w/o SB	
ID	Execution Time	H	Execution Time	H
1	**0.07**	**8**	0.08	8
2	**0.11**	**9**	0.13	9
3	**0.19**	**10**	0.26	10
4	**0.25**	**11**	0.30	11
5	**0.34**	**12**	0.59	12
6	**0.56**	**13**	0.66	13
7	**0.62**	**14**	0.66	14
8	**0.79**	**15**	0.77	15
9	**0.84**	**16**	1.25	16
10	**1.36**	**17**	1.34	17
11	**7.41**	**18**	3.46	18
12	**1.85**	**19**	2.46	19
13	**1.87**	**20**	2.14	20
14	2.52	21	**2.48**	**21**
15	**2.57**	**22**	2.64	22
16	5.40	23	**4.43**	**23**
17	6.21	24	**4.78**	**24**
18	**5.67**	**25**	15.83	25
19	**9.03**	**26**	14.79	26
20	10.52	27	**8.13**	**27**
21	22.11	28	**18.31**	**28**
22	**121.28**	**29**	N/A	29
23	**6.7**	**30**	19.25	30
24	**5.8**	**31**	8.69	31
25	38.61	32	**32.66**	**32**
26	14.06	33	**8.96**	**33**
27	**9.76**	**34**	10.01	34
28	14.30	35	**8.83**	**35**
29	**9.92**	**36**	19.20	36
30	**248.48**	**37**	N/A	N/A
31	N/A	N/A	N/A	N/A
32	**207.14**	**39**	N/A	N/A
33	**16.29**	**40**	16.98	40
34	22.38	40	**17.14**	**40**
35	**12.41**	**40**	13.25	40
36	**10.52**	**40**	32.55	40
37	**55.40**	**60**	N/A	N/A
38	N/A	N/A	N/A	N/A
39	N/A	N/A	N/A	N/A
40	N/A	N/A	N/A	N/A

Table 4: Results without rotation, with and without symmetry breaking constraints.

As shown, the algorithm using Breaking Symmetry resolves more number of instances in a little while way by millisecond. So, It seems that such a implementation of symmetry propagation with to minor changes on the original model allows to integrate the acceleration mechanisms. but in general in our cases we can say that the performance of the 2 Model is quite the same as same value of H(or L) is achieved in the solved instance and specially for the first 30 instance where the obtained plates are square shaped with an obtained H equal to the given W. As assuming before that the objective value; interested in is the minimum plate height achieved between Models in order to mark the optimality, and as Shown that in most case where the instance is solved in the 2 Models, the achieved Height(H) is equal, that why we tend to highlight the optimal solution in such cases by taking the fast Execution time.

By keeping the Breaking symmetry constraints, also the rotation model fails to solve more instances comparing to the previous model (without rotation) and especially the last complex instances composed of 25 circuits and above, Overall, the model with rotation achieves slightly worse results, managing to solve less instances and taking more time to find an optimal solution for most of them but aside of that the Objective value of H is quite the same in the 2 Models for the first 30 instances and it differs in the last 10 instances and this is highlighted in the below table from instance 30 and above . But as mentioned before the solving time without rotation was better and this is marked also in the following graphical presentation of the solving time for all instances with and without Rotation.

	Objective value: H	
ID	**SAT + SB**	**SAT + SB + Rotation**
30	**37**	**37**
31	N/A	**38**
32	**39**	**39**
33	**40**	N/A
34	40	N/A
35	**40**	N/A
36	**40**	N/A
37	**60**	N/A
38	N/A	N/A
39	N/A	N/A
40	N/A	N/A

Table 5: Optimality results for the last 10 instances with and without rotation.

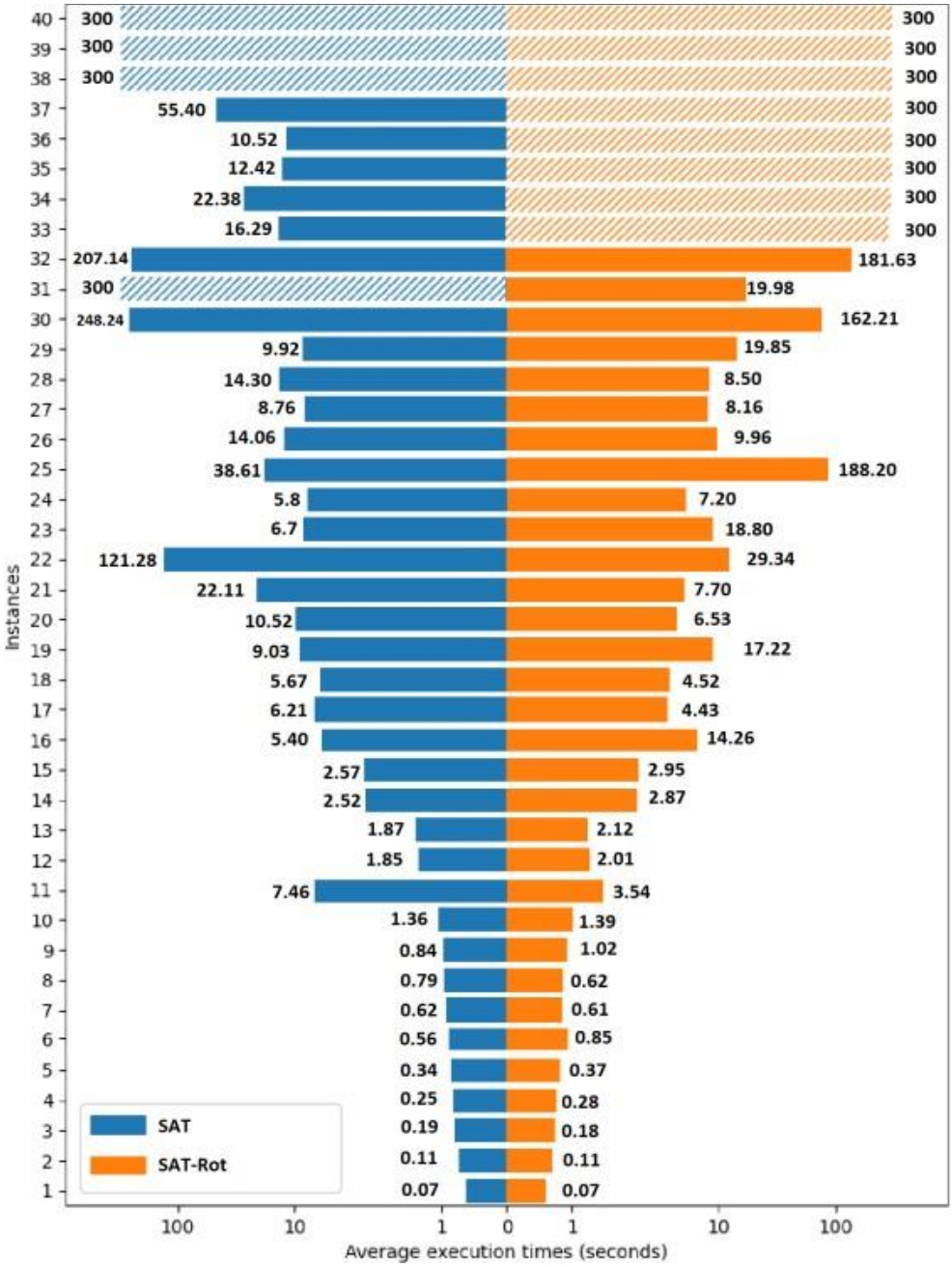


Figure 3: Graphical comparison of the solving time between rotation model and no rotation model.

The implemented SAT model as shown actually managed to solve the optimality of 30 to 32 out of 40 instances within the given time constraint. We were interested in developing the SAT approach to compare its results with the CP solvers developed before and as its looks this solver has small improvement in the solving performance of VLSI problem except the last 5 instances of large numbers of integrated circuits. Due to that we are going to deal with the MIP Model in the next part in order to get a conclusion concerning the best Model that solve VLSI.

4 SMT Model

This part is going to address the implementation of the SATisfiability Modulo Theories approach to the VLSI problem. In the following sections we will present the decision variables and the objective function implemented to solve this problem. Moreover, we will define the constraints used for this project, in addition to the experimental design and results.

4.1 Decision variables

In order to approach the Very Large Scale Integration problem we need to define several decision variables. It has been noticed that this problem is very similar to the two-dimensional bin packing one, especially its strip packing variation, in which the objective is to pack without overlap a set I of two-dimensional rectangular items into a strip of a finite width W but infinite height H in order to minimize the height used [2]. Therefore, we need to define several decision variables to take into account the positions of the circuits and the height of the circuit.
In order to consider the optimal positions of a set of circuits inside a fixed-width plate, we need two arrays to store the x and y coordinates, respectively *xPos* and *yPos*, for each circuit i in the current instance. Every decision variable inside the aforementioned arrays can assume a value between 0 and an upper bound which is given by $w - min(x)$ for the coordinates on the x-axis and by $maxLength - min(y)$ for the coordinates on the y-axis, where $min(x)$ and $min(y)$ defines respectively the circuit with the shortest width and the one with the shortest length, while w is the width of the plate for each instance i, and *maxLength* is the upper bound of the objective function, as described in the following section 4.2.

$$0 \leq xPos_i \leq w - min(x), \qquad i \in [1, numberOfCircuits] \tag{3}$$

$$0 \leq yPos_i \leq maxLength - min(y), \qquad i \in [1, numberOfCircuits] \tag{4}$$

4.2 Objective function

The objective function is to minimize L, that is the length of the plate. To facilitate the work of the solver, we defined the lower and upper bounds of the objective function as follow:

- The lower bound is obtained when the plate is completely fitted and there aren't more spaces between the circuits

$$area_i = x_i * y_i$$

$$lowBound = \frac{\sum_{i=1}^{n} (area_i)}{w}$$

- The upper bound is obtained when all the circuits are stacked in one single column

$$maxLength = upBound = \sum_{i=1}^{n} (y_i) \tag{5}$$

Anyhow, we noticed how the upper bound does not particularly influence the performances of the solver, this is because it searches mainly for values near the lower bound, since it is what we want to minimize.

4.3 Constraints

Firstly, we need to define some constraints to reduce the search space. The most obvious thing to implement is the domain constraint of the decision variables, that is the coordinates of the circuits, as

showed in Equations 3, 4. Therefore we added to the model the two following constraints:

$$\bigwedge_{i=1}^{n} 0 \leq xPos_i \leq w - min(x)$$

$$\bigwedge_{i=1}^{n} 0 \leq yPos_i \leq max\_length - min(y)$$

The first one indicates that the position of the circuit on the x-axis can't be outside of the maximum width of the whole plate, similarly the position of the circuit on the y-axis can't be outside the maximum length, defined in Equation 5, of the plate.
Another very important constraint is the one that makes sure that two different circuits don't overlap each other. In fact we do not want that, since each circuit should be able to occupy its own space on the plate. The mentioned constraint has been implemented in the following way:

$$\bigvee_{i,j \in [1,n] | i<j} (xPos_i + x_i \leq xPos_j) \vee (xPos_j + x_j \leq xPos_i) \vee (yPos_i + y_i \leq yPos_j) \vee (yPos_j + y_j \leq yPos_i)$$

In fact, it is enough just for one condition to stand, therefore a logical OR between four different conditions has been implemented.

4.3.1 Implied Constraints

In addition to the constraints just mentioned, we noticed that this problem can be seen as a bi-dimensional scheduling problem. In fact, we can see the plate width as the time axis, the plate height as the required resources and the circuits as tasks. Specifically, the coordinate of the circuit on the x-axis represents the starting time of the task, the width its duration and the height its resource requirements. Therefore, we enforced a cumulative constraint in our model as follows:

$$\sum_{i=1 | xPos_i \leq u < xPos_i + x_i} y_i \leq max\_length \qquad u \in [1, w]$$

4.3.2 Symmetry breaking constraints

In order to speed up the search for a feasible and hopefully optimal solution, we implemented some symmetry breaking constraints.
Firsty, we positioned the bigger circuit in the origin coordinates of the plate, as follows:

$$x_i Pos[maxArea] = 0$$

$$y_i Pos[maxArea] = 0$$

Moreover, we also studied the case for rectangles of the same size:

$$lex\leq ([x_1, y_1], [x_2, y_2]), \qquad (w1 = w2) \vee (h1 = h2)$$

4.4 Rotation Model

We are also requested to handle the case in which it is possible to consider the possibility for the circuits to be rotated. To distinguish a rotated circuit from another one which is not rotated, we need to define the following variable:

$$rotation_i = \begin{cases} 1 \text{ if circuit is rotated} \\ 0 \text{ otherwise} \end{cases} \qquad \forall\ i \in [1, numberOf Circuits]$$

In this case, we can define the rotation of a circuit as follows:

$$\bigwedge_{i=1} (rotation_i) \rightarrow ((x_i = y_i) \wedge (y_i = x_i))$$

$$\bigwedge_{i=1} (\neg rotation_i) \rightarrow ((x_i = x_i) \wedge (y_i = y_i))$$

Having defined this, we can now replace with *xRot* and *yRot* the (x, y) decision variables defined earlier:

$$xRot_i + (rot_i * y_i) + (1 - rot_i) * x_i \leq w \qquad i \in [1, numberOf Circuits]$$

$$yRot_i + (rot_i * x_i) + (1 - rot_i) * y_i \leq maxLength \qquad i \in [1, numberOf Circuits]$$

This means that when $rot_i = 1$ for the circuit i, we will take into account for the x-coordinates the length with respect to the y-axis and viceversa. At the same time, the non-overlapping constraints become:

$$xRot_i + (rot_i * y_i) + (1 - rot_i) * x_i \leq xRot_j$$
$$xRot_j + (rot_j * y_j) + (1 - rot_j) * x_j \leq xRot_i$$
$$yRot_i + (rot_i * x_i) + (1 - rot_i) * y_i \leq yRot_j$$
$$yRot_j + (rot_j * x_j) + (1 - rot_j) * y_j \leq yRot_i$$

Regarding the symmetry breaking constraints for the rotation model, we added to the ones identified in Section 4.3.2 the following one:

$$\bigwedge_{i=1}^{n} (x_i = y_i) \rightarrow \neg rotation_i$$

That is, if a circuit has the same width and the same height, therefore it is pointless to apply a rotation to it.

4.5 Validation

Before presenting the experimental results, we will first briefly go through the architecture we used to implement the SMT model and the experimental design.
We decided to use pySMT [3], a Pyhton API that provides an intermiedate step between SMT-LIB and solvers API. In this way, we could build a model without it being solver-specific.
Since pySMT doesn't support optimization, we implemented an algorithm to iterate the solving process over different maximum lengths starting from the lower bound, doing this way we are sure to meet an optimal solution.

Algorithm 1 Optimization

```
h ← l_min
while h ≤ max_length ∧ ¬solver_is_sat do
    h ← h + 1
    solver.set(Constraints, Equals(h, h_min)
end while
```

4.5.1 Experimental design

PySMT can leverage the API of the following solvers:

- MathSAT
- Z3

- CVC4
- Yices 2
- CUDD
- PicoSAT
- Boolector

Between the ones above mentioned, only the first four can manage integers variables. Anyway, we chose to test only Z3 and CVC4.
Z3, also known as Z3 Theorem Prover, is a cross-platform satisfiability modulo theories solver developed by Microsoft.
CVC4 is an efficient open-source automatic theorem prover for satisfiability modulo theories problems. It can be used to prove the validity (or, dually, the satisfiability) of first-order formulas in a large number of built-in logical theories and their combination.
Anyway, we managed to get partially good results only using Z3, for this reason, for the sake of brevity, the CVC4 results have been left out.

4.5.2 Experimental results

In this section, we compared the performances and the results obtained by the solvers using different techniques. Firstly, we tested the model using the Z3 solver with symmetry breaking constraints for both the rotation and no rotation model. The results are showed in Table 6.

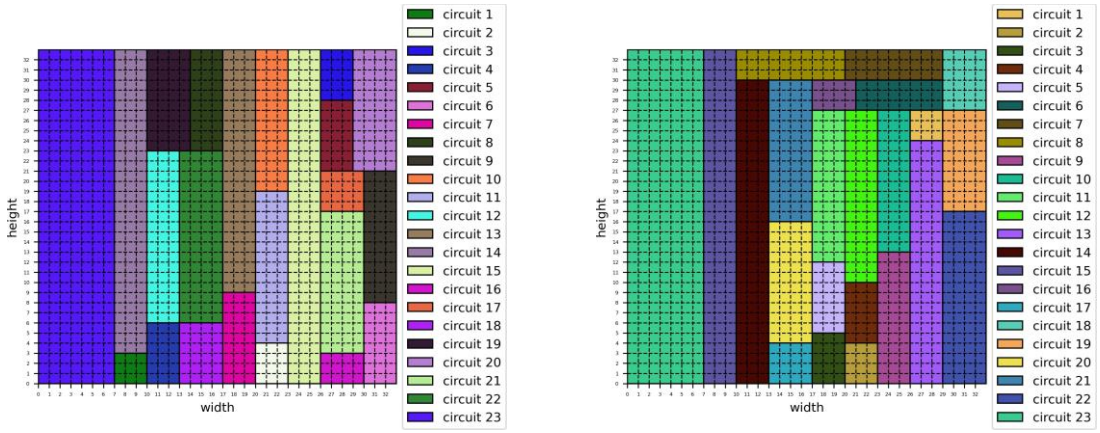
ID	L	Time (s)
1	**12**	**0.070**
2	**9**	**0.016**
3	**10**	**0.019**
4	**11**	**0.022**
5	**12**	**0.027**
6	**13**	**0.032**
7	**14**	**0.050**
8	**15**	**0.037**
9	**16**	**0.036**
10	**17**	**0.073**
11	**18**	**2.081**
12	**19**	**0.165**
13	**20**	**0.219**
14	**21**	**0.982**
15	**22**	**0.088**
16	**23**	**4.861**
17	**24**	**2.546**
18	**25**	**6.613**
19	**26**	**8.250**
20	**27**	**4.293**
21	**28**	**26.771**
22	**29**	**66.146**
23	**30**	**2.509**
24	**31**	**1.715**
25	**32**	**175.496**
26	**33**	**108.240**
27	**34**	**5.605**
28	**35**	**9.540**
29	**36**	**22.167**
30	N/A	300
31	**38**	**0.264**
32	N/A	300
33	**40**	**2.589**
34	**40**	**125.601**
35	40	167.509
36	**40**	3.761
37	N/A	300
38	N/A	300
39	**60**	**68.963**
40	N/A	300

(a) Without rotation

ID	L	Time (s)
1	12	0.159
2	9	0.018
3	10	0.029
4	11	0.064
5	12	0.119
6	13	0.136
7	14	0.169
8	15	0.141
9	16	0.118
10	17	0.212
11	18	3.827
12	19	2.554
13	20	0.707
14	21	3.286
15	22	1.645
16	23	68.284
17	24	7.946
18	25	10.229
19	26	94.516
20	N/A	300
21	28	98.472
22	N/A	300
23	30	31.908
24	31	25.434
25	N/A	300
26	33	283.781
27	34	11.127
28	35	32.284
29	N/A	300
30	N/A	300
31	38	40.236
32	N/A	300
33	40	44.145
34	N/A	300
35	40	50.317
36	40	71.708
37	N/A	300
38	N/A	300
39	N/A	300
40	N/A	300

(b) With rotation

Table 6: Results using Z3 with symmetry breaking constraints. L is the value of the solution found. N/A means that no solution has been found for that particular instance.



(a) 26-ins without rotation (b) 26-ins with rotation

Figure 4: Solutions to the instance n.26 with and without rotation.

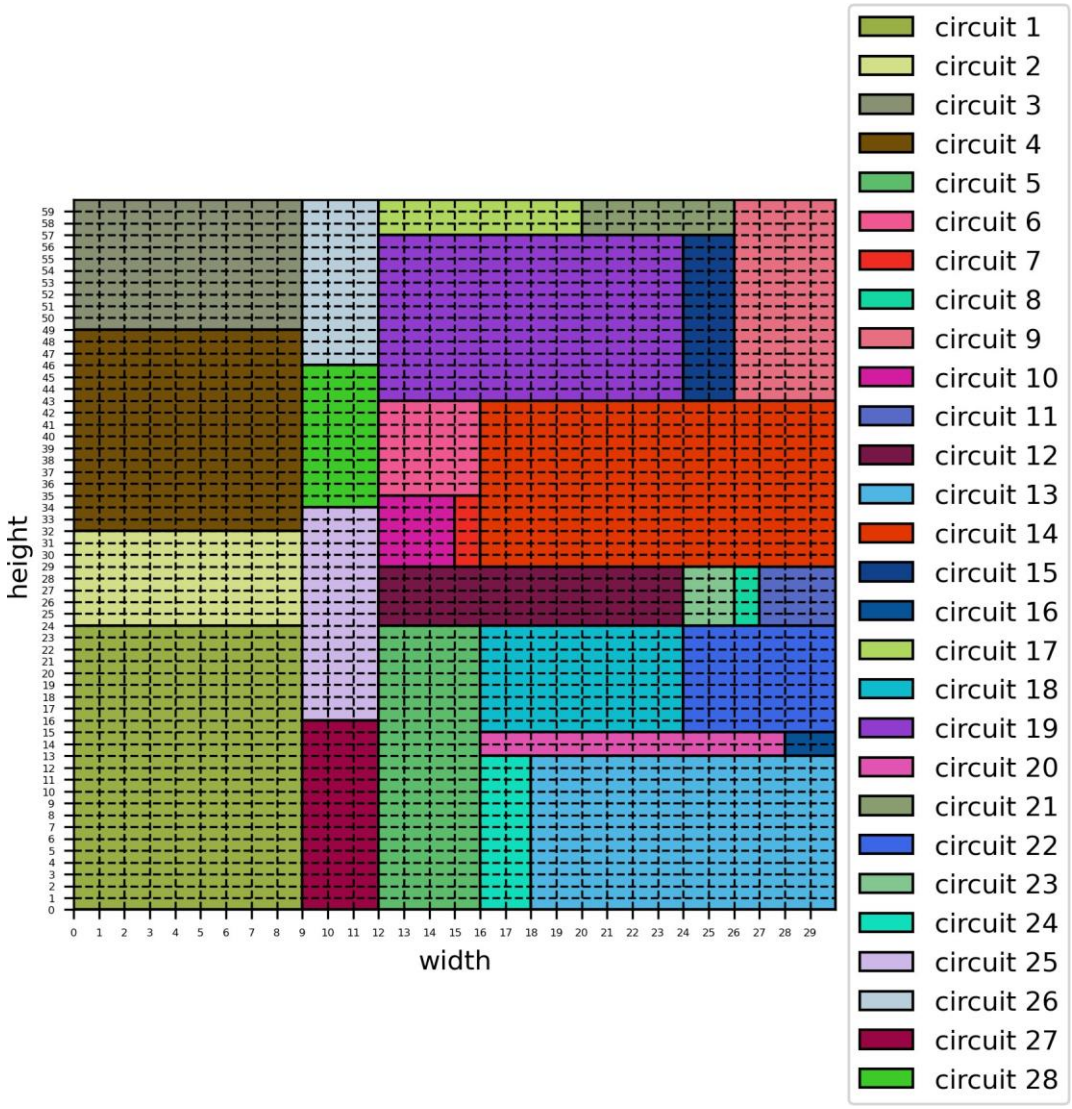


Figure 5: Result of the instance n.39 with symmetry breaking constraints and with no rotation.

It has been taken into account also the variation with no symmetry breaking constraint, leading to the following results:

ID	L	Time (s)
1	**12**	**0.047**
2	**9**	**0.013**
3	**10**	**0.017**
4	**11**	**0.024**
5	**12**	**0.041**
6	**13**	**0.038**
7	**14**	**0.057**
8	**15**	**0.029**
9	**16**	**0.037**
10	**17**	**0.057**
11	**18**	**2.587**
12	**19**	**0.086**
13	**20**	**0.275**
14	**21**	**1.097**
15	**22**	**0.286**
16	**23**	**66.621**
17	**24**	**0.952**
18	**25**	**0.514**
19	**26**	**21.138**
20	**27**	**15.217**
21	**28**	**15.959**
22	N/A	300
23	**30**	**1.244**
24	**31**	**10.481**
25	N/A	300
26	**33**	**68.359**
27	**34**	**9.761**
28	**35**	**10.054**
29	**36**	**21.808**
30	N/A	300
31	**38**	**0.898**
32	N/A	300
33	**40**	**0.328**
34	N/A	300
35	**40**	**184.972**
36	**40**	**14.791**
37	N/A	300
38	N/A	300
39	N/A	300
40	N/A	300

(a) Without rotation

ID	L	Time (s)
1	**12**	**0.060**
2	**9**	**0.018**
3	**10**	**0.029**
4	**11**	**0.055**
5	**12**	**0.080**
6	**13**	**0.191**
7	**14**	**0.129**
8	**15**	**0.651**
9	**16**	**0.202**
10	**17**	**1.584**
11	**18**	**9.672**
12	**19**	**0.242**
13	**20**	**1.571**
14	**21**	**1.253**
15	**22**	**5.132**
16	**23**	**16.644**
17	**24**	**13.751**
18	**25**	**78.675**
19	**26**	**86.024**
20	N/A	300
21	N/A	300
22	N/A	300
23	**30**	**41.597**
24	**31**	**52.459**
25	N/A	300
26	**33**	**13.029**
27	**34**	**213.053**
28	**35**	**105.996**
29	**36**	**218.937**
30	N/A	300
31	**38**	12.064
32	N/A	300
33	**40**	**10.818**
34	N/A	300
35	**40**	**123.898**
36	**40**	**40.198**
37	N/A	300
38	N/A	300
39	N/A	300
40	N/A	300

(b) With rotation

Table 7: Results using pySMT without symmetry breaking constraints.

5 MIP Model

A Mixed Integer Programming problem is a mathematical optimization program in which some decision variables are restricted to be integers, and the remaining ones are not discrete. In this section, we will define a model following the aforementioned approach.

5.1 Decision variables

The decision variables for this model are the same defined in Section 4.1 for the SMT model.

5.2 Objective function

As with the decision variables, also the objective function can be found in Section 4.2 for the SMT model.

5.3 Constraints

In order to solve the problem, we have to pose several constraints. In fact we want to make sure that all the circuits stay inside the bounds of the plate, while at the same time not overlapping each other and minimizing the total length. So, firstly we introduced a constraint to control the bounds of each circuit:

$$x_i + xPos_i \leq w, \qquad i \in [1, numberOfCircuits]$$

$$y_i + yPos_i \leq maxLength, \qquad i \in [1, numberOfCircuits]$$

As mentioned above, we need to implement some overlapping constraints in order to make sure the coordinates of each circuit inside the plate don't overlap each other when placed inside it. To implement this constraint, we want to make sure that at least one of the following equations hold:

$$xPos_i + w_i \leq xPos_j, \quad \text{or}$$

$$xPos_j + w_j \leq xPos_i, \quad \text{or}$$

$$yPos_i + h_i \leq yPos_j, \quad \text{or}$$

$$yPos_j + h_j \leq yPos_i,$$

Since in Mixed Integer Programming it doesn't exist any explicit OR implementation, this condition can be modeled with the help of binary variables δ and big-M constraints [4], in the following way:

$$xPos_i + w_i \leq xPos_j + M_1\delta_{i,j,1}$$
$$xPos_j + w_j \leq xPos_i + M_2\delta_{i,j,2}$$
$$yPos_i + h_i \leq yPos_j + M_3\delta_{i,j,3}$$
$$yPos_j + h_j \leq yPos_i + M_4\delta_{i,j,4}$$
$$\sum_k(\delta_{i,j,k}) \leq 3$$

The binary variables make sure at least one of the constraints is active. It is important to make the constraints M_i as small as possible, in this case we set $M1 = M2 = W$ and $M3 = M4 = maxLength$. The i and j subscripts indicates two different circuits, this will be explained in the next subsection 5.3.1.

5.3.1 Symmetry breaking constraints

In order to reduce the search space of the solving algorithm, we tried to implement some symmetry breaking constraints.
Since we are dealing with different circuits, we need to compare multiple circuits, i.e. we generate

these combinations $\forall i, j$. Anyway, we can't compare a circuit with itself. A first approach could have been to compute the constraints for $i = j$ but we can apply an optimization. Namely, if rectangles i and j are not overlapping we don't need to check the pair of circuits j and i, that is we only need constraints and variables $\sum_k(\delta_{i,j,k})$ for $\boldsymbol{i < j}$.
Furthermore, it has been noticed that different symmetric solutions can be obtained by changing the position of the biggest circuit, for this reason it has been enforced the circuit with the biggest area to be in the coordinates (0,0).

$$x_i Pos[maxArea] = 0$$
$$y_i Pos[maxArea] = 0$$

5.4 Rotation

We also need to handle the case in which it is possible for the circuits to be rotated inside the plate, which means that an $w \times h$ circuit can be positioned either as it is or as $h \times w$. To do that we had to define another model with some minor changes.
Initially, in order to distinguish a rotated circuit from another one which is not we need to add to the model another set of binary variables:

$$rot_i = \begin{cases} 1 \text{ if circuit is rotated} \\ 0 \text{ otherwise} \end{cases} \qquad \forall\ i \in [1, numberOf Circuits]$$

Having defined this, we can now replace with $xRot$ and $yRot$ the (x, y) decision variables defined earlier:

$$xRot_i + (rot_i * y_i) + (1 - rot_i) * x_i \leq w \qquad i \in [1, numberOf Circuits]$$
$$yRot_i + (rot_i * x_i) + (1 - rot_i) * y_i \leq maxLength \qquad i \in [1, numberOf Circuits]$$

This means that when $rot_i = 1$ for the circuit i, we will take into account for the x-coordinates the length with respect to the y-axis and viceversa. At the same time, the non-overlapping constraints become:

$$xRot_i + (rot_i * y_i) + (1 - rot_i) * x_i \leq xRot_j + M_1\delta_{i,j,1}$$
$$xRot_j + (rot_j * y_j) + (1 - rot_j) * x_j \leq xRot_i + M_2\delta_{i,j,2}$$
$$yRot_i + (rot_i * x_i) + (1 - rot_i) * y_i \leq yRot_j + M_3\delta_{i,j,3}$$
$$yRot_j + (rot_j * x_j) + (1 - rot_j) * y_j \leq yRot_i + M_4\delta_{i,j,4}$$
$$\sum_k(\delta_{i,j,k}) \leq 3$$

The same symmetry breaking constraints applied for the standard model hold also for the rotation one.

5.5 Validation

5.5.1 Experimental design

We used PuLP [5], a LP modeler written in Python, in order to solve the problem. A very convenient peculiarity of this Python API is that it makes possible to define a problem and to use different optimizers to solve it without changing the code. Therefore, we tested the model using three different optimizers: CPLEX [6], Gurobi [7] and the one offered by PuLP:

- **CPLEX**: IBM ILOG CPLEX Optimization Studio is an optimization software package. PuLP offers a CPLEX LP/MIP solver via Python binding. Anyway, it is also possible to use the official solver, via the full path of the executable. We decided to choose the latter because it supports additional options to pass to the solver.

- **Gurobi**: The Gurobi Optimizer is a state-of-the-art solver for mathematical programming. Unfortunately, we could not get the full license, as a consequence we had to use the free license which allowed us to solve only some instances, as we will see in the experimental results' section 5.5.2.

- **PuLP CBC**: This solver used a precompiled version of CBC, that is the COIN-OR Branch and Cut solver which is an open-source mixed-integer programming solver written in C++.

Regarding CPLEX, as mentioned above, we set the following options for the solver:

- **Preprocessing symmetry = 5**: This option describes whether symmetry breaking reductions will be automatically executed during the preprocessing phase. Level 5 exerts an extremely aggressive level of symmetry breaking.

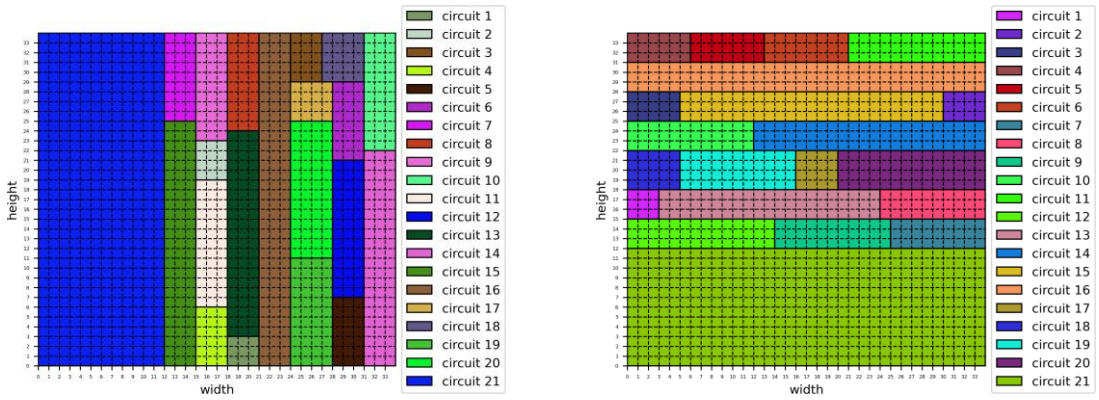
- **Threads = 8**: It sets the default number of parallel threads that will be invoked by any CPLEX parallel optimizer.

- **MIP Strategy Probe = 3**: It sets the amount of probing on variables to be performed before MIP branching. Level 3 defines a very aggressive probing level.

- **MIP Emphasis Switch = 2**: It controls the trade-off between speed, feasibility and moving best-bounds. A value equals to 2 emphasizes optimatility over feasibility.

5.5.2 Experimental results

In conclusion, we compared the performances and the results obtained by the solvers using different techniques.
In Table 8 we showed the results we got using the CPLEX solver. We managed to solve to optimality 31 instances out of 40 using the model without rotation and 29 instances with rotation. On the other hand, both the rotation and the non rotation model fail to solve the most complex instances, especially the last one composed of 73 different circuits. Overall, the model with rotation achieves slightly worse results, managing to solve less instances and taking more time to find an optimal solution for most of them. In Figure 6 it is showed the 27th instance without rotation and with rotation.



(a) 27-ins without rotation (b) 27-ins with rotation

Figure 6: Solutions to the instance n.27 with and without rotation.

ID	L	Time (s)	Status
1	12	0.031	1
2	9	0.032	1
3	10	0.041	1
4	11	0.043	1
5	12	0.074	1
6	13	0.083	1
7	14	0.088	1
8	15	0.089	1
9	16	0.131	1
10	17	0.182	1
11	18	1.601	1
12	19	0.367	1
13	20	0.379	1
14	21	1.564	1
15	22	0.889	1
16	23	14.356	1
17	24	2.696	1
18	25	4.991	1
19	26	48.644	1
20	27	8.405	1
21	28	22.878	1
22	29	189.032	1
23	30	17.243	1
24	31	7.994	1
25	33	300	0
26	33	16.452	1
27	34	15.438	1
28	35	10.391	1
29	36	60.632	1
30	38	300	0
31	38	1.135	1
32	41	300	0
33	40	12.938	1
34	41	300	0
35	41	300	0
36	40	12.081	1
37	61	300	0
38	62	300	0
39	61	300	0
40	103	300	0

(a) Without rotation

ID	L	Time (s)	Status
1	12	0.074	1
2	9	0.043	1
3	10	0.048	1
4	11	0.065	1
5	12	0.078	1
6	13	0.079	1
7	14	0.118	1
8	15	0.093	1
9	16	0.162	1
10	17	0.597	1
11	18	8.903	1
12	19	2.418	1
13	20	1.960	1
14	21	1.346	1
15	22	8.786	1
16	23	170.245	1
17	24	5.246	1
18	25	133.524	1
19	27	300	0
20	28	300	0
21	29	300	0
22	30	300	0
23	30	43.864	1
24	31	14.244	1
25	33	300	0
26	33	122.560	1
27	34	34.368	1
28	35	138.105	1
29	36	121.570	1
30	38	300	0
31	38	18.482	1
32	41	300	0
33	40	8.190	1
34	40	105.671	1
35	40	205.492	1
36	40	183.148	1
37	61	300	0
38	62	300	0
39	61	300	0
40	118	300	0

(b) With rotation

Table 8: Results using CPLEX with symmetry breaking constraints. Status = 1 means an optimal solution has been found; status = 0 means an optimal solution has not been found due to the time constraint. L is the value of the solution found.

The difference in the computational time for solving the given instances with the two main model can be seen in Figure 7. From this graph, it is even more clear how the rotation model presents an increased difficulty with respect to the standard one. In fact, some instances like the 19th, the 20th, the 21st and the 22nd, which the standard model can solve fairly easily, can't be solved in the given time by CPLEX if rotation of the circuits has to be taken into account.

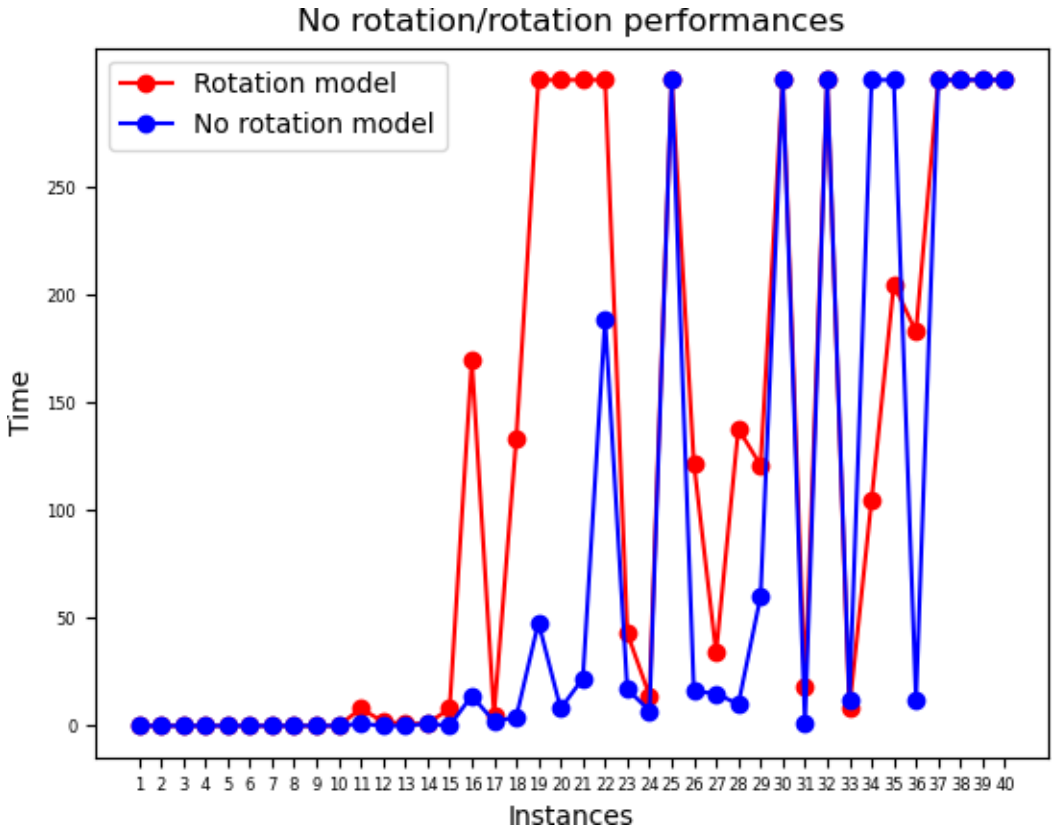


Figure 7: Graphical comparison of the difference between rotation model and no rotation model.

As briefly mentioned above, the free licence we managed to obtain from Gurobi did not allow us to successfully solve all the instances. For this reason, only the first 31 instances have been shown in Table 9. It failed only two instances without rotation and seven instances with rotation. Generally, the results obtained by Gurobi are better than CPLEX, as we will see later with a global comparison. In Figure 8 is displayed the last instance solved by Gurobi. Instead, in Figure 9 there is a comparison between the instance 16 with and without rotation.

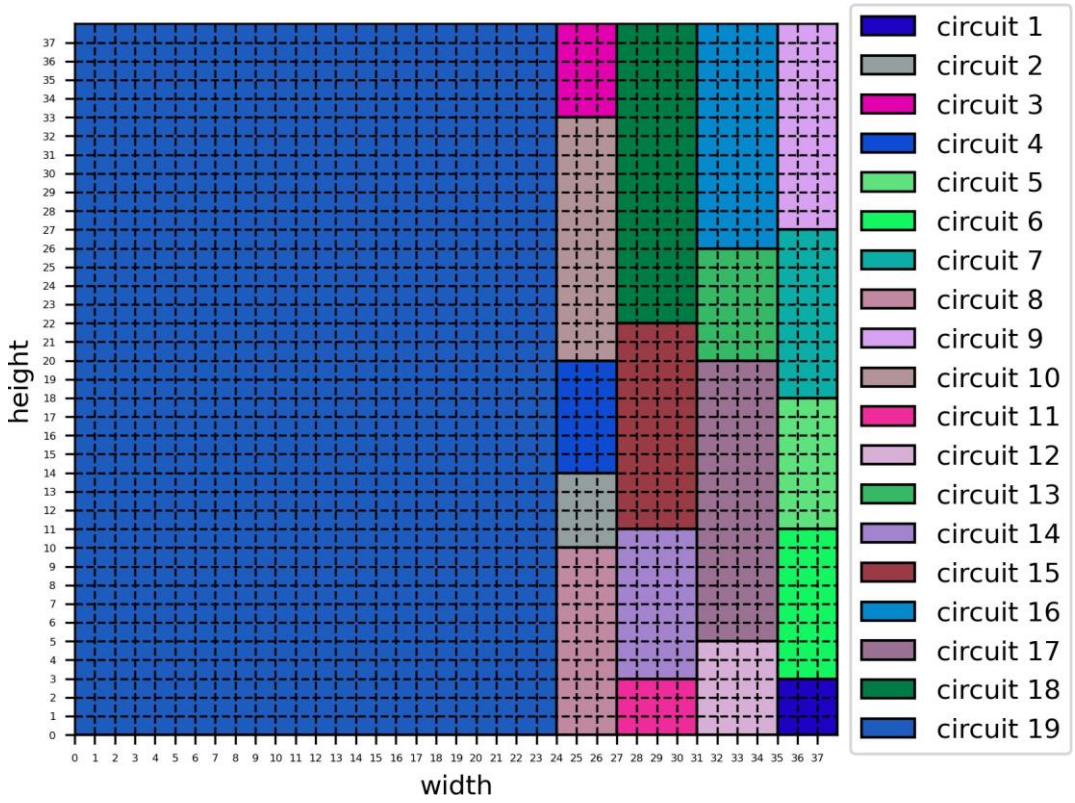
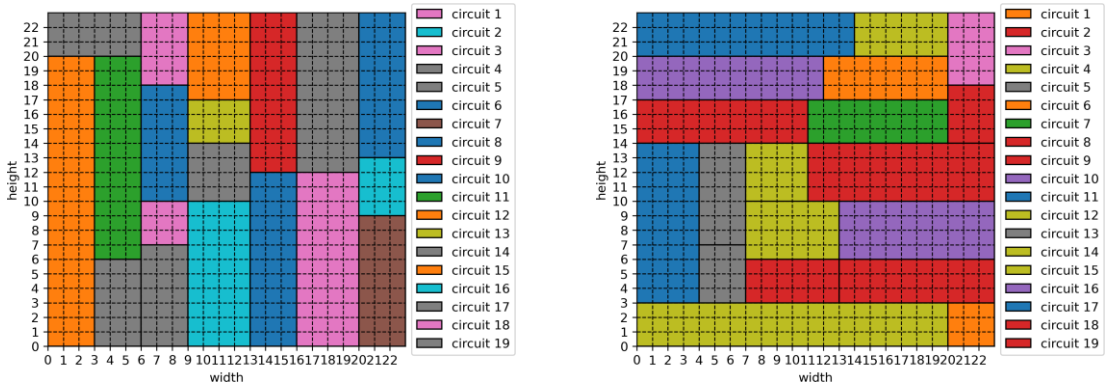


Figure 8: 31st instance solved using Gurobi



(a) 16-ins without rotation

(b) 16-ins with rotation

Figure 9: Solutions to the instance n.16 with and without rotation computed with Gurobi.

The PuLP CBC solver offered by the PuLP Python API has been found to not be very good compared to the other ones already mentioned in this report. In fact, it failed to find an optimal solution in time for most of the instances, managing to solve to optimality only 10 instances out of 40 with no rotation, and 9 with rotation.

For this reason, in Table 10 is displayed the results obtained only for the first instances. It is worth mentioning how even the computational time is higher than the corresponding instances for the other solvers. This is already noticeable from the 10th instance solved without rotation, which took more than one minute to solve, opposed to less than one second for CPLEX and Gurobi.

In Figure 10 we showed a histogram displaying the solving times for the no rotation problem of each solver mentioned so far, for the first 31 instances. As already said, it is highly noticeable how

ID	L	Time (s)	Status
1	12	0.001	1
2	9	0.001	1
3	10	0.001	1
4	11	0.015	1
5	12	0.031	1
6	13	0.031	1
7	14	0.016	1
8	15	0.001	1
9	16	0.016	1
10	17	0.015	1
11	18	0.318	1
12	19	0.239	1
13	20	1.321	1
14	21	0.314	1
15	22	1.203	1
16	23	1.816	1
17	24	0.298	1
18	25	1.899	1
19	26	11.611	1
20	27	8.058	1
21	28	5.139	1
22	30	300	0
23	30	3.008	1
24	31	1.177	1
25	32	36.874	1
26	33	15.929	1
27	34	1.126	1
28	35	3.498	1
29	36	3.731	1
30	38	300	0
31	38	0.283	1

(a) Without rotation

ID	L	Time (s)	Status
1	12	0.001	1
2	9	0.001	1
3	10	0.001	1
4	11	0.016	1
5	12	0.052	1
6	13	0.066	1
7	14	0.126	1
8	15	0.141	1
9	16	0.047	1
10	17	0.314	1
11	18	2.809	1
12	19	8.140	1
13	20	0.674	1
14	21	2.589	1
15	22	4.815	1
16	23	87.137	1
17	24	43.952	1
18	26	300	0
19	27	300	0
20	28	300	0
21	29	300	0
22	30	300	0
23	30	16.670	1
24	31	12.272	1
25	33	300	0
26	34	300	0
27	34	20.712	1
28	35	173.377	1
29	36	25.449	1
30	38	300	0
31	38	9.026	1

(b) With rotation

Table 9: Results using Gurobi with symmetry breaking constraints

PuLP CBC is the worst one, taking the maximum given time for 20 instances.

ID	L	Time (s)	Status
1	12	0.581	1
2	9	0.059	1
3	10	0.126	1
4	11	1.498	1
5	12	2.184	1
6	13	1.235	1
7	14	1.398	1
8	15	3.806	1
9	16	20.994	1
10	17	72.624	1

(a) Without rotation

ID	L	Time (s)	Status
1	12	0.049	1
2	9	0.033	1
3	10	0.059	1
4	11	0.069	1
5	12	1.118	1
6	13	41.961	1
7	14	14.332	1
8	15	28.463	1
9	16	61.339	1
10	18	300	0

(b) With rotation

Table 10: Results using PuLP_CBC with symmetry breaking constraints

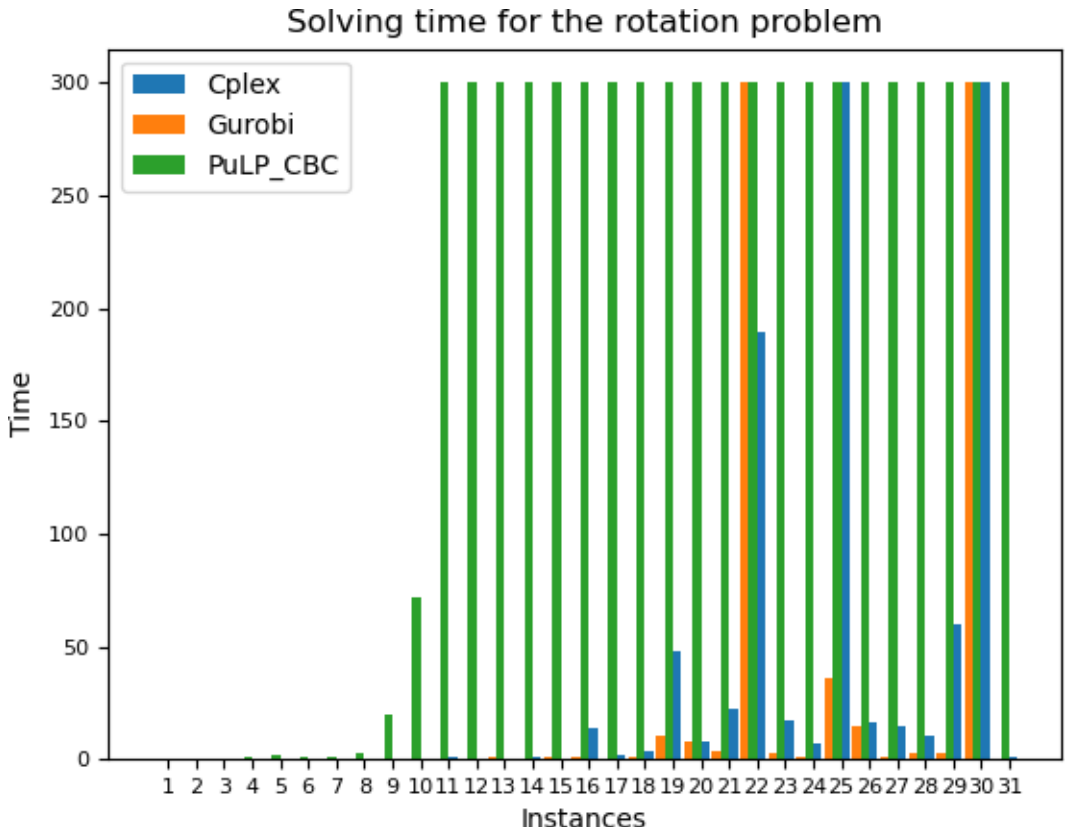


Figure 10: Comparison of the solving time for the no rotation problem of different solvers.

Afterwards, we proceeded to test the model without symmetry breaking constraints. Namely, we removed the CPLEX option setting for the preprocessing symmetry and also the constraints described in the specific section 5.3.1.
In Table 11 we can see the results for this strategy compared to the optimal solutions found with the symmetry breaking constraints. The results in bold are optimal solutions, N/A means a solution could not be found by the solver. We can notice how the results obtained without taking into account the symmetries of the problem tend to be worse than the first approach. For example, Gurobi without symmetry breaking could solve to optimality only 26 instances, opposing to the 29 solved with it.

ID	CPLEX	Gurobi	PuLP_CBC	CPLEX w/o SBC	Gurobi w/o SBC	PuLP_cbc w/o SBC
1	**12**	**12**	**12**	**12**	**12**	**12**
2	**9**	**9**	**9**	**9**	**9**	**9**
3	**10**	**10**	**10**	**10**	**10**	**10**
4	**11**	**11**	**11**	**11**	**11**	**11**
5	**12**	**12**	**12**	**12**	**12**	**12**
6	**13**	**13**	**13**	**13**	**13**	**13**
7	**14**	**14**	**14**	**14**	**14**	**14**
8	**15**	**15**	**15**	**15**	**15**	**15**
9	**16**	**16**	**16**	**16**	**16**	**16**
10	**17**	**17**	**17**	**17**	**17**	**17**
11	**18**	**18**	37	**18**	**18**	56
12	**19**	**19**	24	**19**	**19**	39
13	**20**	**20**	41	**20**	**20**	44
14	**21**	**21**	23	**21**	**21**	23
15	**22**	**22**	23	**22**	**22**	23
16	**23**	**23**	25	**23**	**23**	25
17	**24**	**24**	27	**24**	**24**	27
18	**25**	**25**	27	**25**	**25**	28
19	**26**	**26**	30	27	27	29
20	**27**	**27**	30	**27**	**27**	29
21	**28**	**28**	32	**28**	**28**	33
22	**29**	30	32	**29**	**29**	35
23	**30**	**30**	32	**30**	**30**	38
24	**31**	**31**	34	**31**	**31**	34
25	33	**32**	41	33	33	39
26	**33**	**33**	35	34	34	36
27	**34**	**34**	37	**34**	**34**	38
28	**35**	**35**	40	**35**	**35**	38
29	**36**	**36**	39	**36**	37	39
30	38	38	42	**37**	38	41
31	**38**	**38**	39	**38**	**38**	41
32	41	N/A	44	40	N/A	44
33	**40**	N/A	41	**40**	N/A	43
34	41	N/A	45	41	N/A	48
35	41	N/A	44	**40**	N/A	45
36	**40**	N/A	45	**40**	N/A	44
37	61	N/A	69	62	N/A	67
38	62	N/A	74	62	N/A	73
39	61	N/A	75	61	N/A	69
40	103	N/A	N/A	103	N/A	310

Table 11: Results obtained with CPLEX, Gurobi an PuLP_CBC with and without symmetry breaking constraints.

We repeated the same procedure for the rotation problem, finding a similar trend as showed in Table 12. CPLEX without symmetry breaking constraints can solve in the given time one instance less, moreover the computational time increases by a big margin.

ID	CPLEX	Gurobi	PuLP_CBC	CPLEX w/o SBC	Gurobi w/o SBC	PuLP_CBC w/o SBC
1	**12**	**12**	**12**	**12**	**12**	**12**
2	**9**	**9**	**9**	**9**	**9**	**9**
3	**10**	**10**	**10**	**10**	**10**	**10**
4	**11**	**11**	**11**	**11**	**11**	**11**
5	**12**	**12**	**12**	**12**	**12**	**12**
6	**13**	**13**	**13**	**13**	**13**	**13**
7	**14**	**14**	**14**	**14**	**14**	**14**
8	**15**	**15**	**15**	**15**	**15**	**15**
9	**16**	**16**	**16**	**16**	**16**	**16**
10	**17**	**17**	21	**17**	**17**	**17**
11	**18**	**18**	26	**18**	**18**	56
12	**19**	**19**	23	**19**	**19**	39
13	**20**	**20**	28	**20**	**20**	44
14	**21**	**21**	23	**21**	**21**	23
15	**22**	**22**	26	**22**	**22**	23
16	**23**	**23**	25	**23**	**23**	25
17	**24**	**24**	27	**24**	**24**	27
18	**25**	26	29	**25**	**25**	28
19	27	27	30	27	27	29
20	28	28	31	28	**27**	29
21	29	29	34	29	**28**	33
22	30	30	33	30	30	35
23	**30**	**30**	32	**30**	**30**	38
24	**31**	**31**	37	**31**	**31**	34
25	33	33	36	33	33	39
26	**33**	34	37	**33**	34	36
27	**34**	**34**	37	**34**	**34**	38
28	**35**	**35**	38	**35**	36	38
29	**36**	**36**	40	**36**	**36**	39
30	38	38	42	38	38	41
31	**38**	**38**	39	**38**	**38**	41
32	41	N/A	47	41	N/A	44
33	**40**	N/A	45	**40**	N/A	43
34	**40**	N/A	45	41	N/A	48
35	**40**	N/A	45	**40**	N/A	45
36	**40**	N/A	45	**40**	N/A	44
37	61	N/A	71	62	N/A	67
38	62	N/A	69	62	N/A	73
39	61	N/A	73	61	N/A	69
40	116	N/A	N/A	118	N/A	310

Table 12: Results obtained with CPLEX, Gurobi an PuLP CBC with and without symmetry breaking constraints for the rotation problem.

Given the showed results and also the solving time statistics showed in Table 13, we can affirm that the best solver tested is Gurobi. Unluckily as we already mentioned, we could not get the full license of the solver, so we had to use CPLEX to solve all the 40 instances. The performance of the latter, with and without rotation, is showed in Table 14.

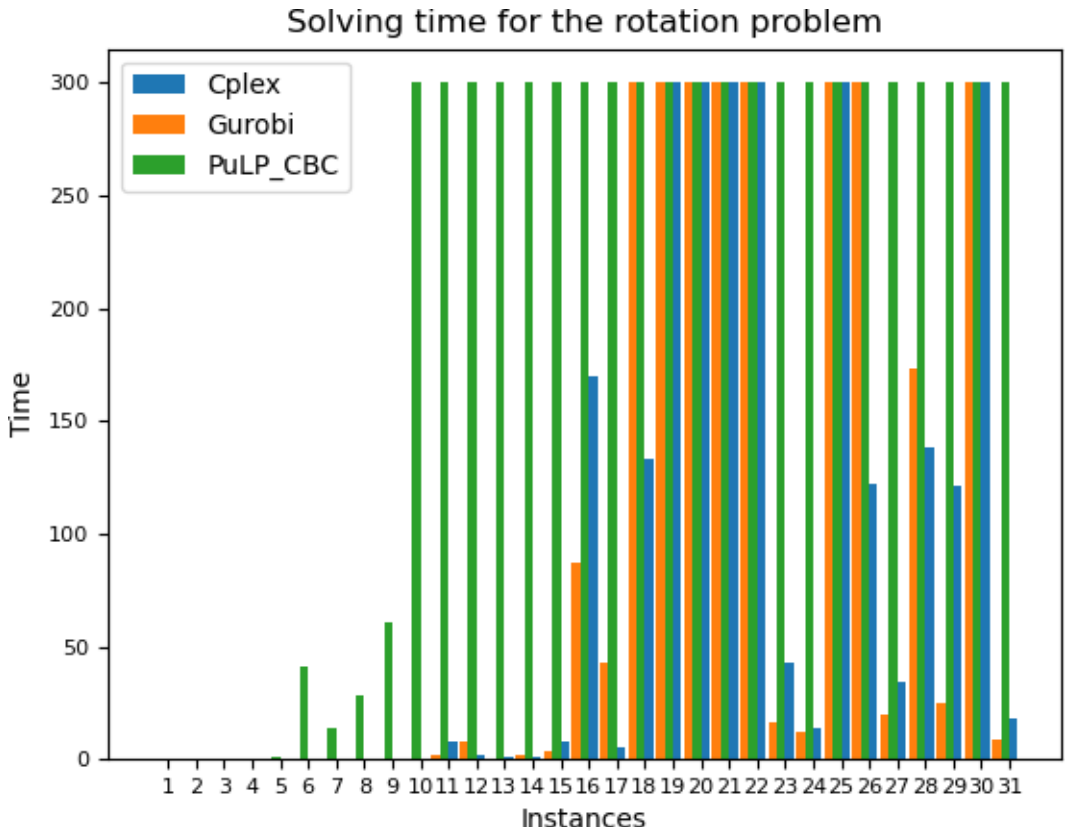


Figure 11: Comparison of the solving time for the rotation problem of different solvers.

Solver	Avg time	Median time	Std time	Found	Optimal
Cplex	33.085	1.6	78.216	31	29
Gurobi	23.261	1.151	74.307	31	29
PuLP-CBC	206.597	300	135.907	31	10

Table 13: Time statistics for the solver with the no rotation problem with symmetry breaking constraints.

Model	Avg time	Median time	Std time	Found	Optimal
Without rotation	78.765	9.395	123.059	40	31
With rotation	115.737	39.116	126.563	40	29

Table 14: Time statistics for CPLEX with and without rotation.

Solver	Avg time	Median time	Std time	Found	Optimal
CPLEX	78.765	9.395	123.059	40	31
CPLEX w/o SBC	88.278	19.541	118.701	40	31

Table 15: Time statistics for CPLEX with rotation and with and without symmetry breaking constraints.

Finally, we showed the instance number 39 solved by CPLEX without rotation and taking into consideration symmetry breaking constraints in Figure 12.

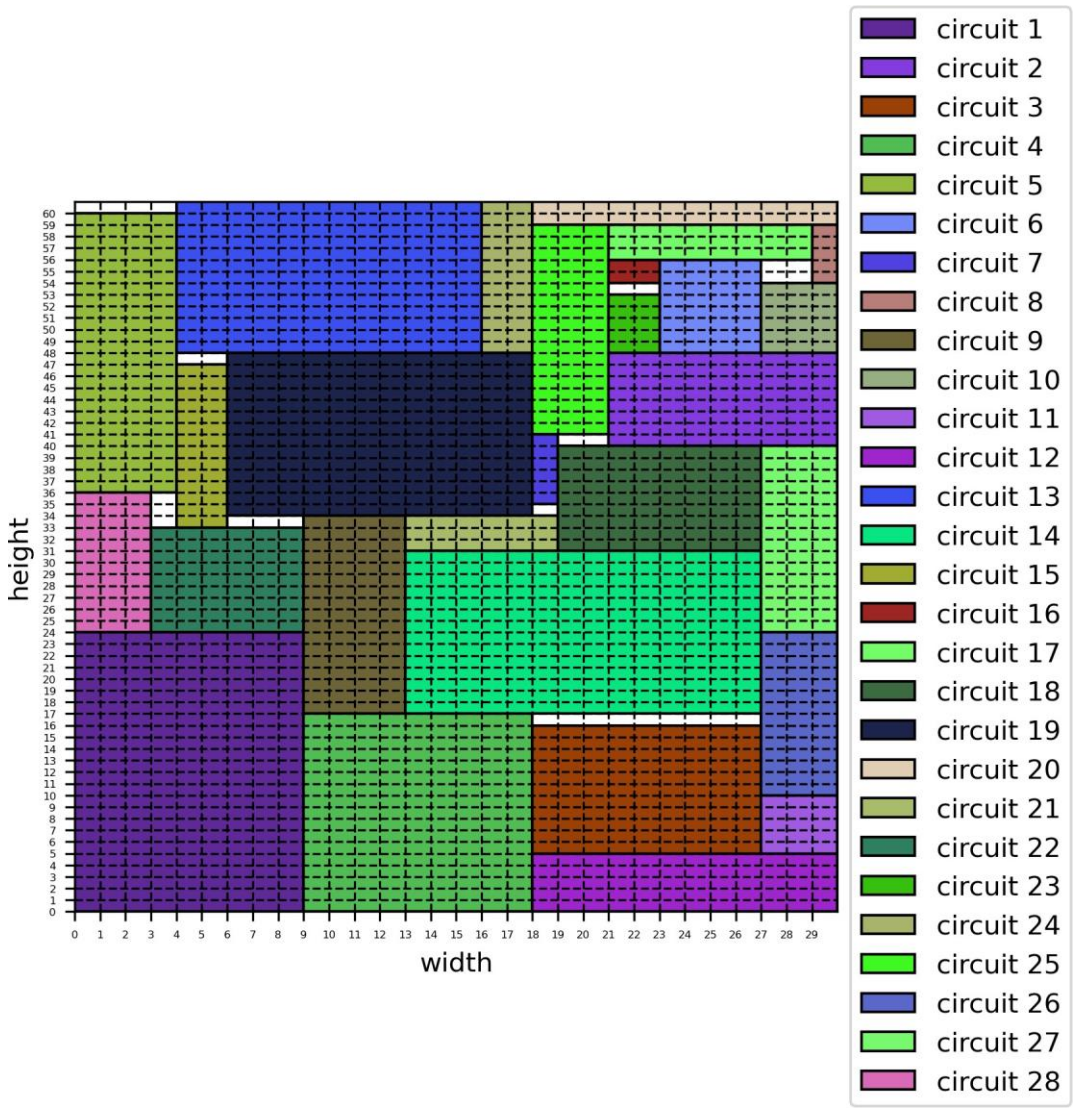


Figure 12: Instance number 39

6 Conclusions

In the final section of this report, we are going to compare all the different models implemented, confronting their runtimes and how they perform for the VLSI problem. First of all, in Table 16 we can see a comparison regarding the approach without rotation and with symmetry breaking constraints implemented.

	CP		SAT		SMT		MIP	
ID	L	Time (s)	L	Time (s)	L	Time (s)	L	Time (s)
1	8	0.4	8	0.1	12	0.1	12	0.0
2	9	0.3	9	0.1	9	0.0	9	0.0
3	10	0.3	10	0.2	10	0.0	10	0.0
4	11	0.3	11	0.3	11	0.0	11	0.0
5	12	0.3	12	0.3	12	0.0	12	0.1
6	13	0.4	13	0.6	13	0.0	13	0.1
7	14	0.3	14	0.6	14	0.1	14	0.1
8	15	0.3	15	0.8	15	0.0	15	0.1
9	16	0.3	16	0.8	16	0.0	16	0.1
10	17	0.5	17	1.4	17	0.1	17	0.2
11	18	108.1	18	7.4	18	2.1	18	1.6
12	19	0.8	19	1.9	19	0.2	19	0.4
13	20	11.8	20	1.9	20	0.2	20	0.4
14	21	1.0	21	2.5	21	1.0	21	1.6
15	22	0.6	22	2.6	22	0.1	22	0.9
16	N/A	300	23	5.4	23	4.9	23	14.4
17	24	274.1	24	6.2	24	2.5	24	2.7
18	25	179.3	25	5.7	25	6.6	25	5.0
19	26	47.5	26	9.0	26	8.3	26	48.7
20	N/A	300	27	10.5	27	4.3	27	8.4
21	N/A	300	28	22.1	28	26.8	28	22.9
22	N/A	300	29	121.3	29	66.1	29	189.0
23	30	197.8	30	6.7	30	2.6	30	17.2
24	31	21.4	31	5.8	31	1.7	31	8.0
25	N/A	300	32	38.6	32	175.5	33	300
26	N/A	300	33	14.0	33	108.4	33	16.5
27	34	50.6	34	9.8	34	5.6	34	15.4
28	N/A	300	35	14.3	35	9.5	35	10.4
29	N/A	300	36	9.9	36	22.2	36	60.6
30	N/A	300	37	248.5	N/A	300	38	300
31	N/A	300	N/A	300	38	0.3	38	1.1
32	N/A	300	39	207.1	N/A	300	41	300
33	N/A	300	40	16.3	40	2.6	40	13.0
34	N/A	300	40	22.4	40	125.6	41	300
35	N/A	300	40	12.4	40	167.6	41	300
36	N/A	300	40	10.5	40	3.8	40	12.1
37	N/A	300	60	55.4	N/A	300	61	300
38	N/A	300	N/A	300	N/A	300	62	300
39	N/A	300	N/A	300	60	69.0	61	300
40	N/A	300	N/A	300	N/A	300	103	300

Table 16: Results taking into account the four different models with no rotation and symmetry breaking constraints implemented.

Similarly, we produced the same table for the rotation model, still using symmetry breaking constraints:

	CP		SAT		SMT		MIP	
ID	L	Time (s)	L	Time (s)	L	Time (s)	L	Time (s)
1	8	0.3	8	0.07	12	0.159	12	0.074
2	9	0.3	9	0.11	9	0.018	9	0.043
3	10	0.4	10	0.18	10	0.029	10	0.048
4	11	0.3	11	0.28	11	0.064	11	0.065
5	12	0.3	12	0.37	12	0.119	12	0.078
6	13	0.4	13	0.85	13	0.136	13	0.079
7	14	7.8	14	0.61	14	0.169	14	0.118
8	15	0.4	15	0.62	15	0.141	15	0.093
9	16	1.3	16	1.02	16	0.118	16	0.162
10	N/A	300	17	1.39	17	0.212	17	0.597
11	N/A	300	18	3.54	18	3.287	18	8.903
12	19	97.5	19	2.01	19	2.554	19	2.418
13	N/A	300	20	2.12	20	0.707	20	1.960
14	N/A	300	21	2.87	21	3.286	21	1.346
15	N/A	300	22	2.95	22	1.645	22	8.786
16	N/A	300	23	14.26	23	68.284	23	170.245
17	N/A	300	24	4.43	24	7.946	24	5.246
18	N/A	300	25	4.52	25	10.229	25	133.524
19	N/A	300	26	17.22	26	94.516	27	300
20	N/A	300	27	6.53	N/A	300	28	300
21	N/A	300	28	7.70	28	98.472	29	300
22	N/A	300	29	29.34	N/A	300	30	300
23	N/A	300	30	18.80	30	31.908	30	43.864
24	N/A	300	31	7.20	31	25.434	31	14.244
25	N/A	300	32	188.20	N/A	300	33	300
26	N/A	300	33	9.96	33	283.781	33	122.560
27	34	28.4	34	8.16	34	11.127	34	34.368
28	N/A	300	35	8.50	35	32.284	35	138.105
29	36	159.5	36	19.85	N/A	300	36	121.570
30	N/A	300	37	162.21	N/A	300	38	300
31	N/A	300	38	19.98	38	40.326	38	18.482
32	N/A	300	39	181.63	N/A	300	41	300
33	40	245.6	N/A	300	40	44.145	40	8.190
34	N/A	300	N/A	300	N/A	300	40	105.671
35	N/A	300	N/A	300	40	50.317	40	205.492
36	N/A	300	N/A	300	40	71.708	40	183.148
37	N/A	300	N/A	300	N/A	300	61	300
38	N/A	300	N/A	300	N/A	300	62	300
39	N/A	300	N/A	300	N/A	300	61	300
40	N/A	300	N/A	300	N/A	300	118	300

Table 17: Results obtained for each different approach using the rotation model.

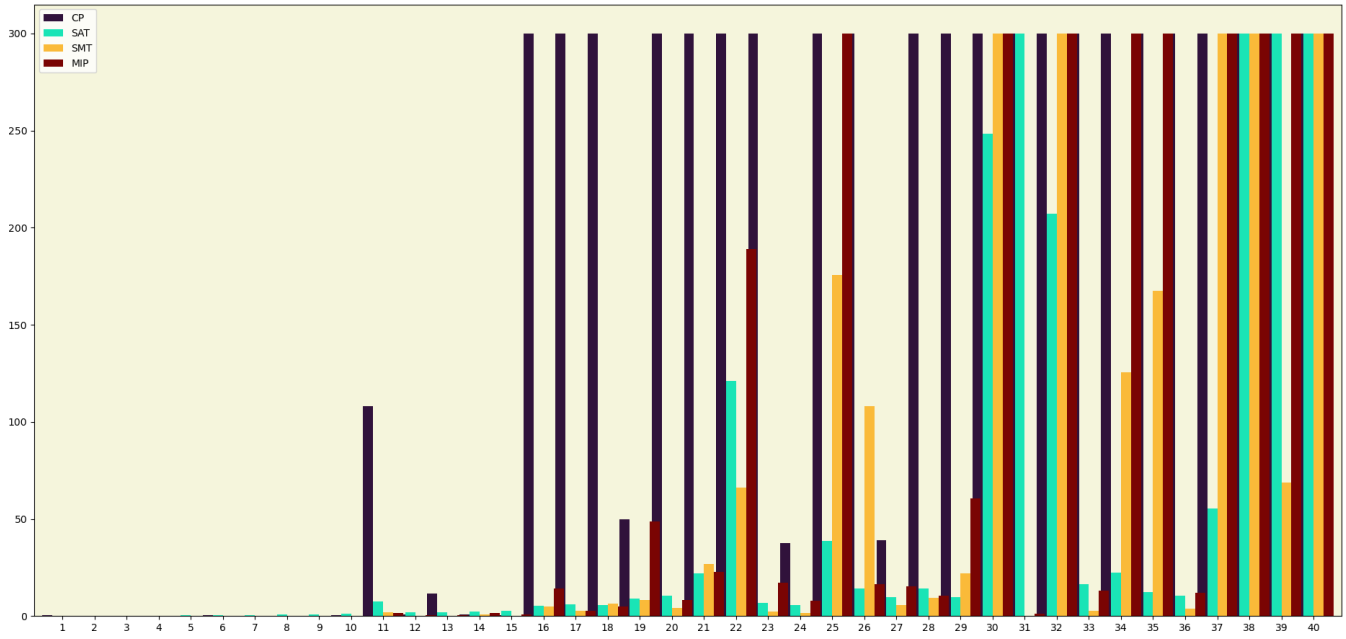


Figure 13: Graphical example of the instance solution given above.

References

[1] J. Petke and P. Jeavons, “The order encoding: From tractable csp to tractable sat,” vol. 6695, pp. 371–372, 06 2011.

[2] A. Neuenfeldt Júnior, *The Two-Dimensional Rectangular Strip Packing Problem*. PhD thesis, 12 2017.

[3] Andrea Michelli, Marco Gario, “pySMT Documentation.” https://pysmt.readthedocs.io/_/downloads/en/latest/pdf/, 2018.

[4] J. P. Vielma, “Mixed integer linear programming formulation techniques,” *Society for Industrial and Applied Mathematics*, vol. 57, no. 1, pp. 3–57, 2015.

[5] COIN-OR Foundation, Inc., “Optimization with PuLP.” https://coin-or.github.io/pulp/, 2022.

[6] IBM, “IBM ILOG CPLEX Optimization Studio CPLEX User’s Manual.” https://www.ibm.com/docs/en/SSSA5P_12.8.0/ilog.odms.studio.help/pdf/usrcplex.pdf, 2022.

[7] Gurobi Optimization, LLC, “Gurobi Optimizer Reference Manual.” https://www.gurobi.com/wp-content/plugins/hd_documentations/documentation/9.0/refman.pdf, 2020.