# VLSI Design Problem

Matteo Nestola matteo.nestola@studio.unibo.it
Simona Scala simona.scala6@studio.unibo.it
Sara Vorabbi sara.vorabbi@studio.unibo.it

## 1 Introduction

Very large-scale integration (VLSI) is the process of integrating or embedding hundreds of thousands of transistors on a single silicon semiconductor microchip. Presently, technologies like smartphones and cellular communications afford unprecedented portability, processing capabilities and application access due to VLSI technology. Because of its attractiveness, we challenged us with this problem.

To solve the VLSI problem, we implemented three different models using Constraint Programming (CP), Satisfiability Modulo Theories (SMT) and Mixed-Integer Linear Programming (MIP).

All the models have in common the following input parameters: $w$, the width of the silicon plate; $n$, the number of necessary circuits to place inside the plate and two vectors, $x$ and $y$, respectively containing the horizontal and vertical dimensions of the circuits. These parameters are provided by the text file of the current instance, processed with the function `read_instance()`. To give an example, we show the values of the first instance:

$$w = 8$$
$$n = 4$$
$$x = [3, 3, 5, 5]$$
$$y = [3, 5, 3, 5]$$

The goal of the problem is to minimize the length of the final device to improve its portability, so in each paradigm we modeled the height of the plate $h$ as the objective variable. Also, we considered two variants of the problem. In the first, 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 $m \times n$. Finally, the outputs are saved as text files and plotted as PNGs with the functions `write_solution()` and `plot_solution()`.

To complete the project it took us four weeks. Each member dealt with the development of one model: in particular, Simona supervised the implementation of CP, Sara was responsible for the development of SMT and Matteo was in charge of MIP implementation. The entire development was quite compelling, thus with some difficulties such as poor documentation and a new programming paradigm based on modeling combinatorial optimization problems.

# 2 CP Model

In Constraint Programming (CP), users declaratively state the constraints on the feasible solutions for a set of decision variables.

## 2.1 Decision variables

The parameters used to define the decision variables have been described in Section 1. The variables of the first variant of the problem are the followings:

- $x_{coord}$, has the domain $[0 \dots x_{cmax}]$ and contains the bottom left x coordinates of the circuits. In particular
$$x_{cmax} = w - \min_{i \in [1 \dots n]} x_i$$
with the meaning that the x coordinate can be at most equal to the difference between the width of the plate and the smallest width of the circuits.

- $y_{coord}$, has the domain $[0 \dots y_{cmax}]$ and contains the bottom left y coordinates of the circuits. In particular
$$y_{cmax} = h_{max} - \min_{i \in [1 \dots n]} y_i$$
with the meaning that the y coordinate can be at most equal to the difference between the maximum height of the plate and the smallest height of the circuits. To better understand the meaning of $h_{max}$ we invite to read the following Section 2.2.

## 2.2 Objective function

In Section 1 we've already mentioned the objective variable of the problem and what is the goal of the objective function. However, in this model $h$ is bounded between $[h_{min} \dots h_{max}]$ where
$$h_{min} = \frac{\sum_{i=1}^{n} x_i \times y_i}{w}$$
with the meaning that a minimum height is obtained when the circuits are perfectly interlocked and occupy a space equal to the sum of their areas, so we can compute it by dividing this area by the width of the plate;

And
$$h_{max} = \sum_{i=1}^{n} y_i$$
with the meaning that the maximum height is the one we get by stacking the circuits on top of each other, so we compute it by adding up the heights of all the circuits.

## 2.3 Constraints

This problem has two global constraints. To reduce blank space inside the plate, we used the `cumulative` constraint which requires that a set of tasks given by start times $s$, durations $d$, and resource requirements $r$, never require more than a global resource bound $b$ at any one time. In this case, we have the following correspondence:

$$s = y_{coord}$$

$$d = y$$

$$r = x$$

$$b = w$$

Since we have to minimize $h$, we use only one `cumulative` constraint on the y axis because the width of the plate is fixed already.

To avoid overlapping between circuits, we used the `diffn` which constrains rectangles $i$, given by their origins $(x_i, y_i)$ and sizes $(dx_i, dy_i)$, to be non-overlapping. The origins are the coordinates $x_{coord}$ and $y_{coord}$ of the circuits and their sizes are their widths and heights, $x$ and $y$.

Another essential constraint is the one that avoids circuits to be placed outside of the plate. We formulated it as follows:

$$\forall i \in [1 \dots n] \qquad x_{coord_i} + x_i \leq w \quad \wedge \quad y_{coord_i} + y_i \leq h$$

**Symmetry breaking constraints** To reduce the horizontal and vertical symmetry of our model we added two extra symmetry breaking constraints. To use them, we first sort the circuits by the size of the area and we store them in the *ordered* vector. The first symmetry breaking constraint requires that the circuit with the largest area has to be placed in the left-bottom area of the plate, such that

$$x_{coord}[C1] < \frac{w}{2} \quad \wedge \quad y_{coord}[C2] < \frac{h}{2}$$

where

$$C1 = ordered[1]$$

$$C2 = ordered[2]$$

So, according to this constraint, the x coordinate of the biggest circuit has to be less than half the width of the plate, while its y coordinate has to be less than half the height of the plate.

The second symmetry breaking constraint requires that the second biggest circuit has to be placed on the right and/or on the top of the biggest one. To define it, we use the lexicographic constraint `lex_lesseq` as follows:

$$(x_{coord}[C1], y_{coord}[C1]) \leq (x_{coord}[C2], y_{coord}[C2])$$

## 2.4 Rotation

The second variant of the problem changes the domains of the previous variables and introduces another decision variable. It follows that:

- The upper bound of $x_{coord}$ becomes

$$x_{cmax} = w - \min\left(\min_{i \in [1 \dots n]} x_i, \min_{i \in [1 \dots n]} y_i\right)$$

  with the meaning that the x coordinate can be at most equal to the difference between the width of the plate and the smallest between the smallest width and height of the circuits.

- The upper bound of $y_{coord}$ becomes

$$y_{cmax} = h_{max} - \min\left(\min_{i\in[1\ldots n]} x_i, \min_{i\in[1\ldots n]} y_i\right)$$

  with the meaning that the y coordinate can be at most equal to the difference between the maximum height of the plate and the smallest between the smallest width and height of the circuits.

- *rotations* is introduced as a binary vector of size $n$ where the i-th element $rotations_i$ is `true` iff the corresponding circuit has been rotated, otherwise it is `false`.

The introduction of the variable *rotations* allows to add an if-then-else statement in the constraints to swap the width and the height of the corresponding circuit

$$rotations_i \rightarrow x_i = y_i \wedge y_i = x_i$$

. Also, we introduced another constraint to avoid the rotation of squared circuits

$$x_i = y_i \rightarrow not(rotations_i)$$

## 2.5 Validation

**Experimental design**

The model was implemented in MiniZincIDE v2.6.4 and it was ran using Chuffed v0.10.4 and Gecode v6.3.0. In addition, we used the library minizinc v0.7.0 to write a Python script to process input and output files, as already discussed in Section 1, and to set the model directly from the shell. In order to reproduce the experiments one can open a terminal in the current working directory `CP` and run the following command

```
python src/cp_exec.py [-f {int}] [-l {int}] [-s {str}] [-sb {True,
False}] [-r {True, False}] [-p {True, False}]
```

- `-f` specifies the number of the first instance
- `-l` specifies the number of the last instance
- `-s` specifies the name of the solver (`'gecode'` or `'chuffed'`)
- `-sb` allows symmetry breaking constraints
- `-r` allows rotation
- `-p` plot the results

Each solver produces four different solutions, depending on the combination between rotation and the use of symmetry breaking constraints provided from the shell.

Trying different combinations of search annotation variables, we've found out that the best approach was searching firstly on $h$ and finally on $x_{coord}$ and $y_{coord}$. The best variable choice annotation for both $x$ and $y$ turned out to be `first_fail`, which means that the solver chooses the variable with the smallest current domain size, while for $h$ we chose `smallest` since it picks the variable with the smallest value in its domain option. Finally, `indomain_min` was set as the way to constraint all three set of variables, which allows to assign them the smallest domain value.

In order to take full advantage of Chuffed's performance, we allowed Chuffed to switch between the defined search and its activity-based search by setting `free search = true`.

The hardware specifications used to run this model are the followings: MacBook Pro, 13-inch, 2018, processor 2,3 GHz Intel Core i5 quad-core, RAM 16 GB 2133 MHz LPDDR3, operating system macOS Ventura 13.0.1.

**Experimental results**

We present the experimental results of this model in the following tables. The first one shows the results of the first variant of the problem, while the second shows the results of the model with rotation allowed.

| ID | Chuffed + SB | Chuffed w/out SB | Gecode + SB | Gecode w/out 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** | UNSAT | UNSAT |
| 11 | **18** | **18** | UNSAT | UNSAT |
| 12 | **19** | **19** | UNSAT | UNSAT |
| 13 | **20** | **20** | UNSAT | UNSAT |
| 14 | **21** | **21** | UNSAT | UNSAT |
| 15 | **22** | **22** | UNSAT | UNSAT |
| 16 | **23** | **23** | UNSAT | UNSAT |
| 17 | **24** | **24** | UNSAT | UNSAT |
| 18 | **25** | **25** | UNSAT | UNSAT |
| 19 | **26** | **26** | UNSAT | UNSAT |
| 20 | **27** | **27** | UNSAT | UNSAT |
| 21 | **28** | **28** | UNSAT | UNSAT |
| 22 | **29** | **29** | UNSAT | UNSAT |
| 23 | **30** | **30** | UNSAT | UNSAT |
| 24 | **31** | **31** | UNSAT | UNSAT |
| 25 | **32** | **32** | UNSAT | UNSAT |
| 26 | **33** | **33** | UNSAT | UNSAT |
| 27 | **34** | **34** | UNSAT | UNSAT |
| 28 | **35** | **35** | UNSAT | UNSAT |
| 29 | **36** | **36** | UNSAT | UNSAT |
| 30 | **37** | **37** | UNSAT | UNSAT |
| 31 | **38** | **38** | UNSAT | UNSAT |
| 32 | **39** | **39** | UNSAT | UNSAT |
| 33 | **40** | **40** | UNSAT | UNSAT |
| 34 | **40** | **40** | UNSAT | UNSAT |
| 35 | **40** | **40** | UNSAT | UNSAT |
| 36 | **40** | **40** | UNSAT | UNSAT |
| 37 | **60** | **60** | UNSAT | UNSAT |
| 38 | N/A | N/A | UNSAT | UNSAT |

| | | | | |
|---|---|---|---|---|
| 39 | **60** | **60** | UNSAT | UNSAT |
| 40 | N/A | N/A | UNSAT | UNSAT |

Table 1: Results using Gecode and Chuffed with and without symmetry breaking, considering the variant without rotation

| ID | Chuffed + SB | Chuffed w/out SB | Gecode + SB | Gecode w/out 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** | UNSAT |
| 8 | **15** | **15** | **15** | UNSAT |
| 9 | **16** | **16** | **16** | **16** |
| 10 | **17** | **17** | **17** | **17** |
| 11 | **18** | **18** | **18** | UNSAT |
| 12 | **19** | **19** | UNSAT | UNSAT |
| 13 | **20** | **20** | UNSAT | UNSAT |
| 14 | **21** | **21** | UNSAT | UNSAT |
| 15 | **22** | **22** | UNSAT | UNSAT |
| 16 | **23** | **23** | UNSAT | UNSAT |
| 17 | **24** | **24** | UNSAT | UNSAT |
| 18 | **25** | **25** | UNSAT | UNSAT |
| 19 | **26** | **26** | UNSAT | UNSAT |
| 20 | **27** | **27** | UNSAT | UNSAT |
| 21 | **28** | **28** | UNSAT | UNSAT |
| 22 | N/A | N/A | UNSAT | UNSAT |
| 23 | **30** | **30** | UNSAT | UNSAT |
| 24 | **31** | **31** | UNSAT | UNSAT |
| 25 | **32** | N/A | UNSAT | UNSAT |
| 26 | **33** | **33** | UNSAT | UNSAT |
| 27 | **34** | **34** | UNSAT | UNSAT |
| 28 | **35** | **35** | UNSAT | UNSAT |
| 29 | **36** | **36** | UNSAT | UNSAT |
| 30 | N/A | N/A | UNSAT | UNSAT |
| 31 | **38** | **38** | UNSAT | UNSAT |
| 32 | N/A | N/A | UNSAT | UNSAT |
| 33 | **40** | **40** | UNSAT | UNSAT |
| 34 | **40** | **40** | UNSAT | UNSAT |
| 35 | **40** | **40** | UNSAT | UNSAT |
| 36 | **40** | **40** | UNSAT | UNSAT |
| 37 | N/A | N/A | UNSAT | UNSAT |
| 38 | N/A | N/A | UNSAT | UNSAT |
| 39 | N/A | N/A | UNSAT | UNSAT |
| 40 | N/A | N/A | UNSAT | UNSAT |

Table 2: Results using Gecode and Chuffed with and without symmetry breaking, considering the variant with rotation

For this model we can conclude that the Chuffed solver obtains quite good perfor-

mances which, however, get significantly worse with the Gecode solver. Although we tried different configurations, we weren't able to improve the performances of the model with this solver. We can justify this with the fact that Gecode is faster than Chuffed for very small schedules but tails off much faster.

# 3 SMT Model

Satisfiability modulo theory (SMT) is a problem that is defined through propositional logic. This kind of problem involves variables of different types (integer, real numbers, data structures etc.) and the objective is checking the satisfiability of a formula with respect to a background theory. The implementation for the SMT model was done using the Z3 theorem prover.

## 3.1 Decision variables

The input parameters are already described in Section 1. The decision variables used to define the SMT model are the same as the ones described in Section 2.1. However the formulation of their domain is different, see Section 3.3.

## 3.2 Objective function

Since we were convinced by the structure of the CP model, the objective function of the SMT model is defined as the one of CP mentioned in Section 2.2. We defined the plate height that must be minimized under the name of *min_height*. The optimizer is implemented in Z3.

## 3.3 Constraints

We defined various constraints.

1. **Domain constraints**
   The lower bounds of the coordinates are defined as follow:

$$\begin{aligned} \forall_{i=1}^{n} \quad & x\_coord_i \geq 0 \\ \forall_{i=1}^{n} \quad & y\_coord_i \geq 0 \end{aligned}$$

   The upper bounds are also defined as follow:

$$\begin{aligned} \forall_{i=1}^{n} \quad & x\_coord_i + x\_dim_i \leq w \\ \forall_{i=1}^{n} \quad & y\_coord_i + y\_dim_i \leq min\_height \end{aligned}$$

2. **Non-overlap constraint**
   To avoid the overlapping of the circuits we implemented a constraint that given two blocks $i, j$ makes the placement of the circuits valid if at least one of the following cases is satisfied:

| | |
|---|---|
| $x\_coord_i + x\_dim_i \leq x\_coord_j$ | i to the left of j |
| $y\_coord_i + y\_dim_i \leq y\_coord_j$ | i below j |
| $x\_coord_i - x\_dim_i \geq x\_coord_j$ | i to the right of j |
| $y\_coord_i - y\_dim_i \geq y\_coord_j$ | i above j |

3. **Cumulative constraint**
   This constraint is already described in Section 2.3.

**Symmetry breaking constraints** We added two different symmetry breaking constraint, the first that put the two biggest chips in a lexicographic order and the second, that positioned the biggest chip in the left bottom part of the plate. The mathematical description can be found in Section 2.3.

## 3.4 Rotation

In this variant of the problem we add a new variable to the two previously defined, called *rotation_c*. The definition of this variable is already described in Section 2.4, since the usage in the SMT model is the same as the one in the CP model.

## 3.5 Validation

**Experimental design**

The model was implemented with Python using the Z3 solver v4.11.2.0 from Microsoft Research. To replicate the experiment one can open a terminal in the current working directory `SMT` and run the following command:

```
python src/SMT.py [-f {int}] [-l {int}] [-sb {True, False}]
[-r {True, False}] [-p {True, False}]
```

- `-f` specifies the number of the first instance
- `-l` specifies the number of the last instance
- `-sb` allows symmetry breaking constraints
- `-r` allows rotation
- `-p` plot the results

The hardware specifications used to run this model are the followings: Asus K550J, 15-inch, 2015, processor Intel(R) Core(TM) i7-4720HQ CPU @ 2.60GHz 2.59 GHz, RAM 12.0 GB, operating system Windows 10 Home.

**Experimental results**

We present the experimental results of this model in the following table. As we can see there is a significant decrease in performance in the model that deals with the rotation. Moreover the increasing complexity of the problem does not allow to the model to find solutions for all the instances in the time limit.

| | no_rotation | | rotation | |
|---|---|---|---|---|
| ID | w/out SB | with SB | w/out SB | with 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** | N/A | N/A |
| 11 | N/A | N/A | N/A | N/A |
| 12 | **19** | **19** | N/A | N/A |
| 13 | **20** | **20** | N/A | N/A |
| 14 | **21** | **21** | N/A | N/A |
| 15 | **22** | **22** | N/A | N/A |
| 16 | N/A | N/A | N/A | N/A |
| 17 | **24** | **24** | N/A | N/A |
| 18 | **25** | **25** | N/A | N/A |
| 19 | N/A | N/A | N/A | N/A |
| 20 | N/A | N/A | N/A | N/A |
| 21 | N/A | N/A | N/A | N/A |
| 22 | N/A | N/A | N/A | N/A |
| 23 | **30** | **30** | N/A | N/A |
| 24 | **31** | **31** | N/A | N/A |
| 25 | N/A | N/A | N/A | N/A |
| 26 | N/A | **33** | N/A | N/A |
| 27 | **34** | **34** | N/A | N/A |
| 28 | **35** | N/A | N/A | N/A |
| 29 | **36** | **36** | N/A | N/A |
| 30 | N/A | N/A | N/A | N/A |
| 31 | **38** | **38** | N/A | N/A |
| 32 | N/A | N/A | N/A | N/A |
| 33 | **40** | **40** | N/A | N/A |
| 34 | N/A | N/A | N/A | N/A |
| 35 | N/A | N/A | N/A | N/A |
| 36 | N/A | N/A | N/A | N/A |
| 37 | N/A | N/A | 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 3: Results using SMT Model with and without symmetry breaking considering also the rotation.

# 4 MIP Model

The third method we have used for searching a solution for VLSI design problem is MIP. A Mixed-Integer-Programming problem is one where some of the decision variables are constrained to be integer values at the optimal solution.

## 4.1 Decision variables

The input parameters are already described in Section 1. The decision variables and the formulation of their domains used to define the MIP model are the same as the ones described in Section 2.1. In addition to these, we added $s$, a matrix that contains variables used for the implementation of the big M method (this method will be explained in Section 4.3) The domain of $s$ id defined as follow:

$$s \in B^{n \times n \times 4}$$

where $B := \{0, 1\}$

## 4.2 Objective function

Our objective function is to minimize the height of the plate. In our model the height is represented as a variable $h$, so the goal is to minimize that variable. $h$ is involved into several constraints which have the purpose to reduce its domain to search a solution for the problem. $h$ is strictly related to the information that comes from the instances files, in particular to the chips dimensions. In order to define the domain of $h$, we have defined several parameters:

- $h\_Max$: is the max value for the plate height (computed through the sum of all the value stored in $y$).
- $h\_min$: is the minimal value for the plate height (computed through the sum of all the areas of the chips weighted on $w$).
- $area\_min$: is the sum of all the chips areas.
- $area\_max$: is the product of $w$ and $h\_Max$.

The objective function is already described in Section 2.2.

## 4.3 Constraints

In MIP the constraints must be linear functions. The following rows describes all the constraints involved in our model:

1. **Constraints that verifies if an allocated chip is correctly positioned with respect to plate dimensions $w \cdot h$**

   All the chip must have a width value lower then the width of the plate and a height value lower then the variable h (height of the plate).These constraints are very important and allow all chip allocations outside the plate to be discarded. These constraints are summarize as following:

$$\begin{aligned} x\_cord_i + w\_rotate_i &\leq w & \forall i \in [1..n] \\ y\_cord_i + h\_rotate_i &\leq h & \forall i \in [1..n] \end{aligned}$$

2. **Non-overlap constraints based on the big M method**

   There are basically four cases when considering two chips $i$ and $j$:

$$\begin{aligned} x_i + w_i &\leq x_j && \vee \quad (1) \\ x_j + w_j &\leq x_i && \vee \quad (2) \\ y_i + h_i &\leq y_j && \vee \quad (3) \\ y_j + h_j &\leq y_i && \quad (4) \end{aligned}$$

$$\forall i \in [1..n], \quad \forall j \in [1..n] \quad with \quad i \neq j$$

   All of these cases represent how a chip may be positioned relative to another chip.

   The "or" condition can be modeled with the help of binary variables $s$ and big-M constraints. Each condition is associated to an $s_{ijk}$ binary variable and then is combined to a constant value also called big M value (in our case M assumes two different values:

   - $h$ : in case the constraint is related to the $y$ axis [see (3) and (4) of the formula above]
   - $w$ : in case the constraint is related to the $x$ axis [see (1) and (2) of the formula above]

   The binary variables make sure at least one of the constraints is active (not relaxed). In particular the sum over $k$ of $s_{ijk}$ must be at most equal to 3. The meaning of that is the fact that at least one of the 4 constraints of the no-overlapping for the $i^{th}$ and $j^{th}$ chip must be satisfied.
   So the four constrain above turns into:

$$\begin{aligned} x_i + w_i &\leq x_j + w \cdot s_{ij1} && \text{k=1} \\ x_j + w_j &\leq x_i + w \cdot s_{ij2} && \text{k=2} \\ y_i + h_i &\leq y_j + h \cdot s_{ij3} && \text{k=3} \\ y_j + h_j &\leq y_i + h \cdot s_{ij4} && \text{k=4} \end{aligned}$$

$$\sum_{k=1}^{4} s_{ijk} \leq 3$$

3. **Constraint that check if the area of the plate is bounded by minimal and maximal values**

   To check whether the area occupied by the current chip arrangement is valid, it is necessary to define two constraints that evaluate the area with respect to its maximum and minimum value:

$$\begin{aligned} area &= w * h \\ area &\leq area\_max \\ area &\leq area\_min \end{aligned}$$

   The goal of these constraints is to reduce the blank vertical space.

## 4.4 Rotation

Compared to the model without rotation, it is fundamental to manage a new boolean variable, called *rotation_c*, representing the possibility for a generic chip to be rotated or not. The definition of this variable is already described in Section 2.4 This new variable led to the modification of all constraints that contained the heights and widths of the chips.
Considering a generic chip $i$, if a chip is rotated $rotation\_c[i] = True$, it follows that the width of the chip becomes its new height and the height of the chip becomes its new width.

## 4.5 Validation

**Experimental design**

The model was implemented with Python through the Gurobi Optimizer, a mathematical optimization software library. To replicate the experiment one can open a terminal in the current working directory `MIP` and run the following command:

```
python src/exec_MIP.py [-f {int}] [-l {int}]
[-r {True, False}] [-p {True, False}]
```

- `-f` specifies the number of the first instance
- `-l` specifies the number of the last instance
- `-r` allows rotation
- `-p` plot the results

To enable or disable the symmetry breaking constraints, it's necessary to manually set the symmetry parameter from the .py file as follows:
`model.setParam("Symmetry", value)`
where `value` can be: `-1: auto, 0: off , 1: conservative, 2: aggressive`

The hardware specifications used to run this model are the followings: HP Laptop 15, processor Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz, RAM 16 GB, operating system Windows 11 Home.

**Experimental results**

As we can see from the table summarising the results, the MIP model without rotation produces better results, this is strictly due to the presence of an extra variable to be handled in the case of rotation, which leads to an increase in the complexity of the problem. In fact, as we can see from the table, not all solutions were found in the case of rotation as opposed to the case without rotation.

| | SB = auto | | SB = off | | SB = aggressive | |
|---|---|---|---|---|---|---|
| ID | no_rotation | rotation | no_rotation | rotation | no_rotation | rotation |
| 1 | **8** | **8** | **8** | **8** | **8** | **8** |
| 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** | **18** | **18** | **18** | **18** |
| 12 | **19** | **19** | **19** | **19** | **19** | **19** |
| 13 | **20** | **20** | **20** | **20** | **20** | **20** |
| 14 | **21** | **21** | **21** | **21** | **21** | **21** |
| 15 | **22** | **22** | **22** | **22** | **22** | **22** |
| 16 | **23** | **23** | **23** | **23** | **23** | **23** |
| 17 | **24** | **24** | **24** | **24** | **24** | **24** |
| 18 | **25** | **25** | **25** | **25** | **25** | **25** |
| 19 | 27 | 27 | 27 | **26** | 27 | 27 |
| 20 | **27** | 28 | **27** | 28 | **27** | 28 |
| 21 | 29 | **28** | 29 | 29 | 29 | **28** |
| 22 | **29** | **29** | 30 | **29** | **29** | **29** |
| 23 | **30** | **30** | **30** | **30** | **30** | **30** |
| 24 | **31** | **31** | **31** | **31** | **31** | **31** |
| 25 | 33 | 34 | 33 | 34 | 33 | 34 |
| 26 | 34 | **33** | **33** | **33** | 34 | **33** |
| 27 | **34** | **34** | **34** | **34** | **34** | **34** |
| 28 | **35** | **35** | **35** | **35** | **35** | **35** |
| 29 | **36** | **36** | **36** | **36** | **36** | **36** |
| 30 | 39 | 38 | 38 | 38 | 39 | 38 |
| 31 | **38** | **38** | **38** | **38** | **38** | **38** |
| 32 | 41 | 41 | 40 | 40 | 41 | 41 |
| 33 | **40** | **40** | **40** | **40** | **40** | **40** |
| 34 | 41 | 45 | 41 | 45 | 41 | 45 |
| 35 | 41 | UNSAT | 41 | UNSAT | **40** | UNSAT |
| 36 | **40** | UNSAT | **40** | UNSAT | **40** | UNSAT |
| 37 | 63 | 68 | 62 | 68 | 61 | 68 |
| 38 | 62 | 62 | 62 | 62 | 62 | 62 |
| 39 | 62 | 62 | 61 | 61 | 62 | |
| 40 | 108 | 125 | 107 | 108 | 104 | |

Table 4: Results using MIP Model with various settings for symmetry breaking considering also the rotation.

# 5 Conclusions

We can conclude that the CP model is the best at solving this problem. However, it's the best as long as we use the Chuffed solver: if we use it with the Gecode solver the performances decrease disastrously. Below we show the final table with a row for each instance and a column for each optimization technology we considered.

| | no_rotation | | | rotation | | |
|---|---|---|---|---|---|---|
| ID | CP | SMT | MIP | CP | SMT | MIP |
| 1 | **8** | **8** | **8** | **8** | **8** | **8** |
| 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** | N/A | **16** |
| 10 | **17** | **17** | **17** | **17** | N/A | **17** |
| 11 | **18** | N/A | **18** | **18** | N/A | **18** |
| 12 | **19** | **19** | **19** | **19** | N/A | **19** |
| 13 | **20** | **20** | **20** | **20** | N/A | **20** |
| 14 | **21** | **21** | **21** | **21** | N/A | **21** |
| 15 | **22** | **22** | **22** | **22** | N/A | **22** |
| 16 | **23** | N/A | **23** | **23** | N/A | **23** |
| 17 | **24** | **24** | **24** | **24** | N/A | **24** |
| 18 | **25** | **25** | **25** | **25** | N/A | **25** |
| 19 | **26** | N/A | 27 | **26** | N/A | **26** |
| 20 | **27** | N/A | **27** | **27** | N/A | 28 |
| 21 | **28** | N/A | 29 | **28** | N/A | **28** |
| 22 | **29** | N/A | **29** | N/A | N/A | **29** |
| 23 | **30** | **30** | **30** | **30** | N/A | **30** |
| 24 | **31** | **31** | **31** | **31** | N/A | **31** |
| 25 | **32** | N/A | 33 | **32** | N/A | 33 |
| 26 | **33** | **33** | **33** | **33** | N/A | **33** |
| 27 | **34** | **34** | **34** | **34** | N/A | **34** |
| 28 | **35** | **35** | **35** | **35** | N/A | **35** |
| 29 | **36** | **36** | **36** | **36** | N/A | **36** |
| 30 | **37** | N/A | 38 | N/A | N/A | 38 |
| 31 | **38** | **38** | **38** | **38** | N/A | **38** |
| 32 | **39** | N/A | 40 | N/A | N/A | 40 |
| 33 | **40** | **40** | **40** | **40** | N/A | **40** |
| 34 | **40** | N/A | 41 | **40** | N/A | 45 |
| 35 | **40** | N/A | 41 | **40** | N/A | UNSAT |
| 36 | **40** | N/A | **40** | **40** | N/A | UNSAT |
| 37 | **60** | N/A | 61 | N/A | N/A | 68 |
| 38 | N/A | N/A | 62 | N/A | N/A | 62 |
| 39 | **60** | N/A | 61 | N/A | N/A | 61 |
| 40 | N/A | N/A | 104 | N/A | N/A | 108 |

Table 5: The best result obtained for each instance for the model with rotation