# Combinatorial Decision Making and Optimization Project Work
VLSI Design

Davide Angelani[*], Yellam Naidu Kottavalasa[†], Usha Padma Rachakonda[‡][1]

[1]Department of Computer Science - Informatics and Engineering, Bologna, Italy
[1]Academic Year 2021-2022

[*]davide.angelani@studio.unibo.it
[†]yellam.kottavalasa@studio.unibo.it
[‡]ushapadma.rachakonda@studio.unibo.it

# Contents

## 1 Introduction

The method of building an integrated circuit (IC) by fitting thousands of transistors onto a single chip is know as Very Large-Scale Integration (VLSI). The requirements of today's electronic systems and devices are perfectly suited to VLSI technology. VLSI technology will continue to propel electronics development due to the rising demand for miniaturization, portability, performance, dependability, and functionality.
The main of this project is for a given 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. Here we are applying three different techniques to minimize the final device size: Constraint Programming (CP), Propositional SATisfiability (SAT), Mixed Integer Programming (MIP).

### 1.1 VLSI Instance

An instance of VLSI is a text file consisting of lines of integer values. The first line gives $w$, which is the width of the silicon plate. The following line gives $n$, which is the number of necessary circuits to place inside the plate. Then $n$ lines follow, each with $x_i$ and $y_i$, representing the horizontal and vertical dimensions of the $i$-th circuit.

**Input** Given this input, we want to find a placement of the circuits inside the silicon plate such that the total length of the silicon plate is minimized.

```
13
9
3 3
3 4
3 5
3 6
3 7
3 8
4 3
4 4
7 6
```

**Output** The output of such instance is a text file consisting of lines of integer values. The first lines gives $w$ and $l$, which are the width and the length of the silicon plate. The second line gives $n$, which is the number of circuits placed inside the silicon plate. Then $n$ lines follow, each with $x_i$ $y_i$ $\hat{x}_i$ and $\hat{y}_i$ $rotation_i$, representing the horizontal and vertical dimensions, the bottom left corner and if the circuit is rotated.

```
13 13
9
3 3 3 0 0
3 4 3 3 0
3 5 0 8 0
3 6 10 0 0
```

```
3 7 10 6 0
3 8 0 0 0
4 3 6 0 0
4 4 6 3 0
7 6 3 7 0
```

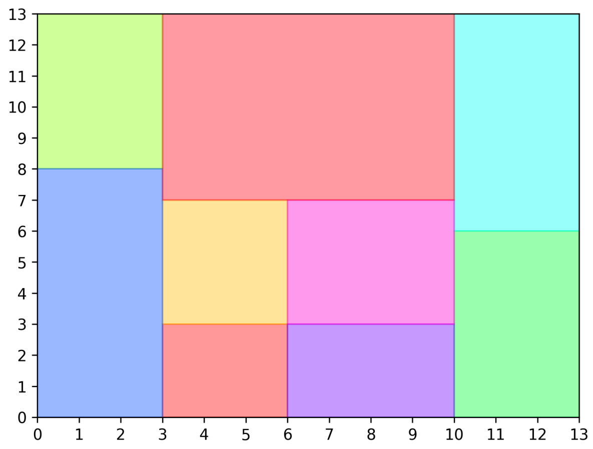

**Figure 1:** VLSI instance output visualized

**Symmetry** The circuits can be placed in various position in the silicon plate and a solution can be reached with various configurations. So to reduce the search space and improve the performance of the model we want to remove such symmetries.
In particular the following symmetries are present in the problem:

- *Horizontal*: two adjacent circuits occupy the same space if they coordinate on the x-axis is swapped.
- *Vertical*: two adjacent circuits occupy the same space if they coordinate on the y-axis is swapped.
- *Rotation*: introduced by rotation, it is similar to the horizontal and vertical symmetries.

## 2 CP Model

Constraint programming is a native satisfiability technology that takes its roots in computer science—logic programming, graph theory, and the artificial intelligence efforts of the 1980s. Recent progress in the development of tunable and robust black-box search for constraint programming engines have turned this technology into a powerful and easy-to-use optimization technology.
Constraint programming is also an efficient approach to solving and optimizing problems that are too irregular for mathematical optimization. This includes time tabling problems, sequencing problems, and allocation or rostering problems.

### 2.1 Decision variables

In order to model the problem the following decision variables are defined:

- $\hat{x}$: with domain [0, $w$], array of size *CIRCUITS*, where each element is the bottom left corner of the circuit in the x-axis.
- $\hat{y}$: with domain [0, *l_max*], array of size *CIRCUITS*, where each element is the bottom left corner of the circuit in the y-axis.
- $l$: with domain [0, *l_max*], is the length of the silicon plate.

## 2.2 Objective function

The objective function is to minimize the length $l$ of the silicon plate. The $l$ is computed as the maximum value of the *p_y* array plus the corresponding height $y$ of the circuit.

$$\min \max_i l = (\hat{y}_i + y_i) \tag{1}$$

## 2.3 Constrains

The defined variables are constrained by the following rules:

- *no overlap* between circuits: the bottom left corner of the circuit $i$ must be greater than the top right corner of the circuit $j$
- *cumulative* constraint: to optimize the occupancy of the silicon plate by the circuits.

Both constraints are ensured by the global constraint of MiniZinc Stuckey, Marriott, and Tack n.d. The first is ensured by the *diffn* constraint.

```
diffn(x_h, y_h, x, y);
```

which given the coordinate of the bottom left corner (*x_h*, *y_h*) and the length and width ($x$, $y$) of the circuit, ensures that the circuit does not overlap with any other circuit.
The second constrain *cumulative* is used in scheduling tasks to enforce the optimization of the allocation of resources.

```
cumulative(s, d, r ,b);
```

In our case the $s$ is the array of the bottom left corner of the circuit in the x-axis, $d$ is the array of the width of the circuit $x$, $r$ is the array of the height of the circuit $y$, and $b$ is the length of the silicon plate $l$. Such constraint is also applied on the y-axis using the array of the bottom left corner of the circuit in the y-axis as $s$, the array of length of the circuit $y$ as $d$, the array of width of the circuit $x$ as $r$ and the width of the silicon plate $w$ as $b$.

**Implied constraints** To improve the performance of the model, we add one implied constraint. Which ensure that the circuits are not placed outside the silicon plate-

$$\max_i (\hat{x}_i + x_i) \leq w \tag{2}$$

**Symmetry breaking constraints** To break the symmetries we noticed that just by restraining the position of the highest circuit we are reducing the number of possible symmetries. So we add the following constrain:

- $\hat{x}_{max} = 0 \wedge \hat{y}_{max} = 0$: the *highest* circuit is placed in the bottom left corner of the silicon plate: in this way we are reducing the search space and the number of possible symmetries

## 2.4 Rotation

The second variant of the problem is to allow the circutis to rotate. This mean that the length and the width of the circuit can be swapped. To tackle this problem we added two array which store the actual length and width of the circuit considering the rotation and an array of boolean variable which indicate if the circuit is rotated or not. The decision variables are the ones described in the section 2.1 plus the following:

- *orientation*: boolean array of size *CIRCUITS*, where each element is true if the circuit is rotated and false otherwise.
- *x_r*: array of size *CIRCUITS*, where each element is the actual width of the circuit considering the rotation. which means that if the circuit $i$ is rotated the $x\_r_i$ is equal to the length $y_i$ of the circuit and vice versa.
- *x_y*: array of size *CIRCUITS*, where each element is the actual length of the circuit considering the rotation. the reverse logic of the *x_r* is applied.

**Objective function** The objective function defined in the section 2.2 is updated accordingly to the rotation using *y_r* instead of $y$.

**Constraints** The constraint are the same of the section 2.3 plus the following:

- $\forall i \in [1, CIRCUITS] : y_i > w \Rightarrow orientation_i = false$: which mean that the circuit cannot be rotated if the length is greater than the width $w$ of the silicon plate.
- $\forall i \in [1, CIRCUITS] : x_i == y_i \Rightarrow orientation_i = false$: symmetry breaking constraint, which mean that the circuit cannot be rotated if the width is equal the length.

## 2.5 Search

MiniZinc allows to define annotations for specifying how to search in order to find a solution to the problem. there are two type of annotations:

- *varchoice*: which specify how to choose the next variable to assign
- *constrainchoice*: which specify a value ordering for the variables

The possible *varchoice* we tested are:

- *input_order*: which assign the variables in the order they are declared in the model
- *dom_wdeg*: which assign the variables based on the weighted degree heuristic

While the possible *constrainchoice* are:

- *indomain_min*: which assign the smallest value to the variable from the domain
- *indomain_random*: which assign the random value to the variable from the domain

Furthemore, MiniZinc allows to define a strategy for restarting the search. The possible restart strategies we tested are:

- *restart_linear*: which restart the search after a fixed number of nodes
- *restart_luby*: which restart the search based on the Luby sequence
- *restart_none*: which do not restart the search

## 2.6 Experiments

### 2.6.1 Experimental design

**SETUP** The experiments are performed on a machine with the following specifications:

- Intel Core i5-8300H (4 core)
- Ram DDR4 8GB
- Nvidia GTX 1050 4GB
- Windows 10 Home x64
- MiniZinc 2.6.4

**Design** For both variant of the problem we define our baseline model as the one without the symmetry breaking constrain and any annotation. Then we have add the constrain and the annotation one by one to see the impact on the performance. We compare the performance of the CP model using two solver.

- gecode: an open-source constraint programming system, which supports many of MiniZinc's global constraints natively
- chuffed: constraint solver based on lazy clause generation

### 2.6.2 Experimental Results for the first variant of the problem

In this section we report the results obtained for the first variant of the problem (without rotation). The tables show the result obtained for each solved instance while the graphs show the time needed to solve the instance. As we can see from the table 1, using the symmetry breaking constraint improve the performance of the solver since we are able to solve more instances. Using any search and restart strategies for both solver worsen the results (tables 2, **??**, **??** and **??**).

| id | chuffed | chuffed + sb | gecode | gecode + sb |
|---|---|---|---|---|
| 1 | 12 | 12 | 12 | 12 |
| 2 | 9 | 9 | 9 | 9 |
| 3 | 10 | 10 | 10 | 10 |
| 4 | 11 | 11 | 11 | 11 |
| 5 | 12 | 12 | 12 | 12 |
| 6 | 13 | 13 | 13 | 13 |
| 7 | 14 | 14 | 14 | 14 |
| 8 | 15 | 15 | 15 | 15 |
| 9 | 16 | 16 | 16 | 16 |
| 10 | 17 | 17 | 17 | 17 |
| 11 | - | 18 | - | 18 |
| 12 | 19 | 19 | 19 | 19 |
| 13 | 20 | 20 | - | 20 |
| 14 | 21 | 21 | 21 | 21 |
| 15 | 22 | 22 | 22 | 22 |
| 17 | 24 | 24 | - | 24 |
| 18 | 25 | 25 | 25 | 25 |
| 20 | - | 27 | - | - |
| 23 | 30 | 30 | - | 30 |
| 24 | - | - | - | 31 |
| 26 | - | 33 | - | - |
| 27 | - | 34 | - | - |
| 28 | - | - | - | 35 |
| 31 | - | 38 | - | 38 |
| 33 | - | - | - | 40 |
| 36 | - | - | - | 40 |

**Table 1:** Results on both solvers with and without symmetry breaking constrain

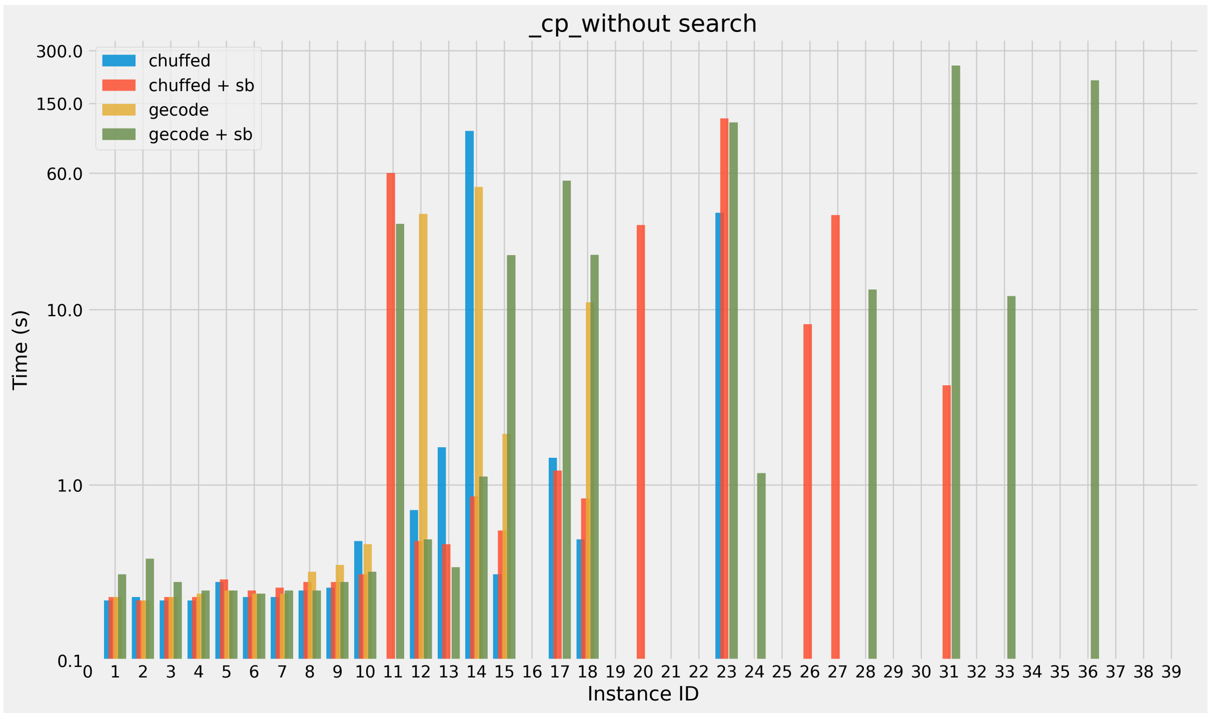

**Figure 2:** Time required to solve the instances for both solvers with and without symmetry breaking constrain

| id | chuffed | gecode |
|---|---|---|
| 1 | 12 | 12 |
| 2 | 9 | 9 |
| 3 | 10 | 10 |
| 4 | 11 | 11 |
| 5 | 12 | 12 |
| 6 | 13 | 13 |
| 7 | 14 | 14 |
| 8 | 15 | 15 |
| 9 | 16 | 16 |
| 10 | 17 | 17 |
| 11 | - | 19 |
| 12 | 19 | 19 |
| 13 | 20 | - |
| 14 | 21 | - |
| 15 | 22 | 22 |
| 17 | 24 | - |
| 18 | 25 | 25 |

**Table 2:** Results obtained on both solver with symmetry breaking constrain and using input_order and indomain_min as annotations

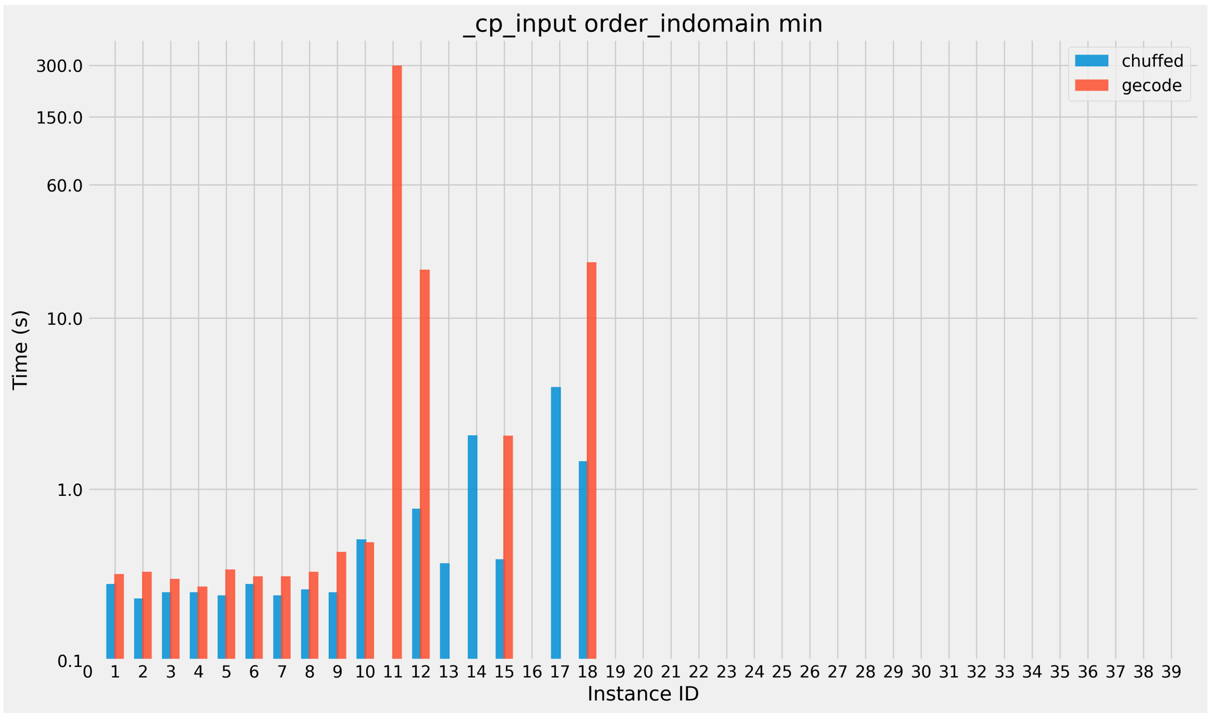

**Figure 3:** Time required to solve the instances for both solvers using input_order and indomain_min as annotations

| id | chuffed | gecode |
|---|---|---|
| 1 | 12 | 12 |
| 2 | 9 | 9 |
| 3 | 10 | 10 |
| 4 | 11 | 11 |
| 5 | 12 | 12 |
| 6 | 13 | 13 |
| 7 | 14 | 14 |
| 8 | 15 | 15 |
| 9 | 16 | 16 |
| 10 | 17 | 17 |
| 12 | 19 | - |
| 13 | 20 | - |
| 14 | 21 | - |
| 15 | 22 | 22 |
| 17 | 24 | - |
| 18 | 25 | - |
| 23 | 30 | - |

**Table 3:** Results obtained on both solver with symmetry breaking constrain and using input_order and indomain_min as annotations with a linear restart

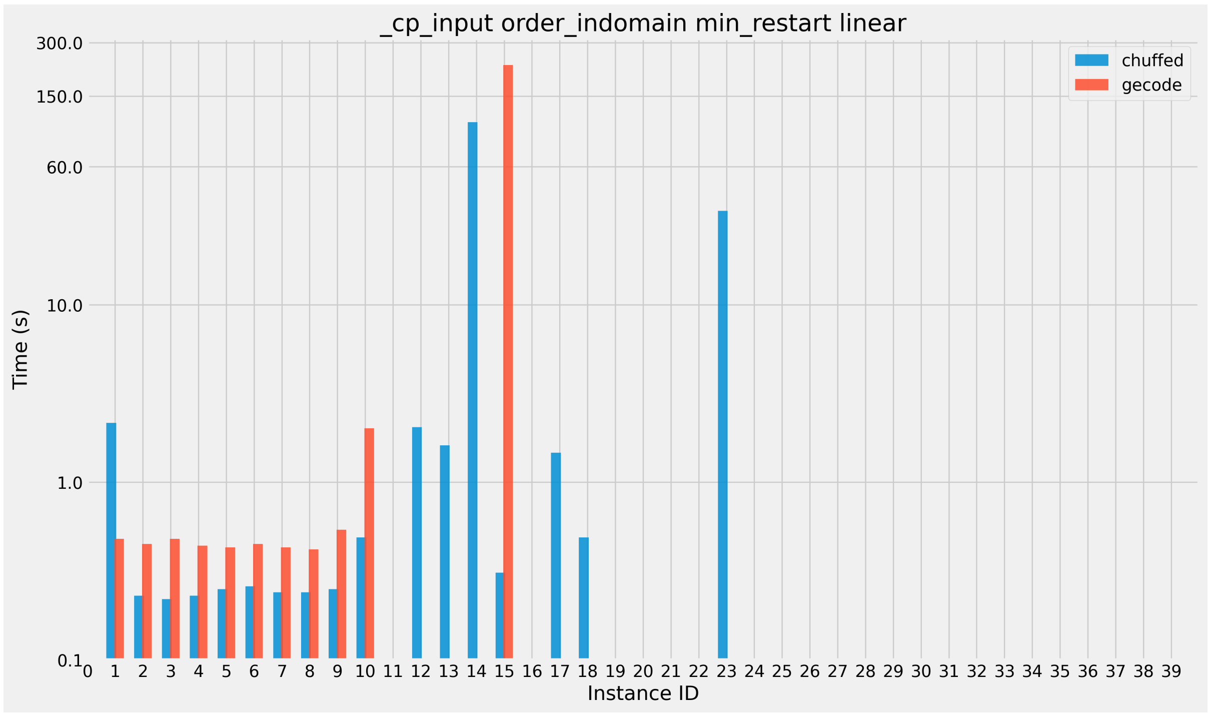

**Figure 4:** Time required to solve the instances for both solvers using input_order and indomain_min as annotations with a linear restart

| id | chuffed | gecode |
|---|---|---|
| 1 | 12 | 12 |
| 2 | 9 | 9 |
| 3 | 10 | 10 |
| 4 | 11 | 11 |
| 5 | 12 | 12 |
| 6 | 13 | 13 |
| 7 | 14 | 14 |
| 8 | 15 | 15 |
| 9 | 16 | 16 |
| 10 | 17 | 17 |
| 12 | 19 | 19 |
| 13 | 20 | - |
| 14 | 21 | - |
| 15 | 22 | 22 |
| 17 | 24 | - |
| 18 | 25 | - |
| 23 | 30 | - |

**Table 4:** Results obtained on both solver with symmetry breaking constrain and using input_order and indomain_min as annotations with a luby restart

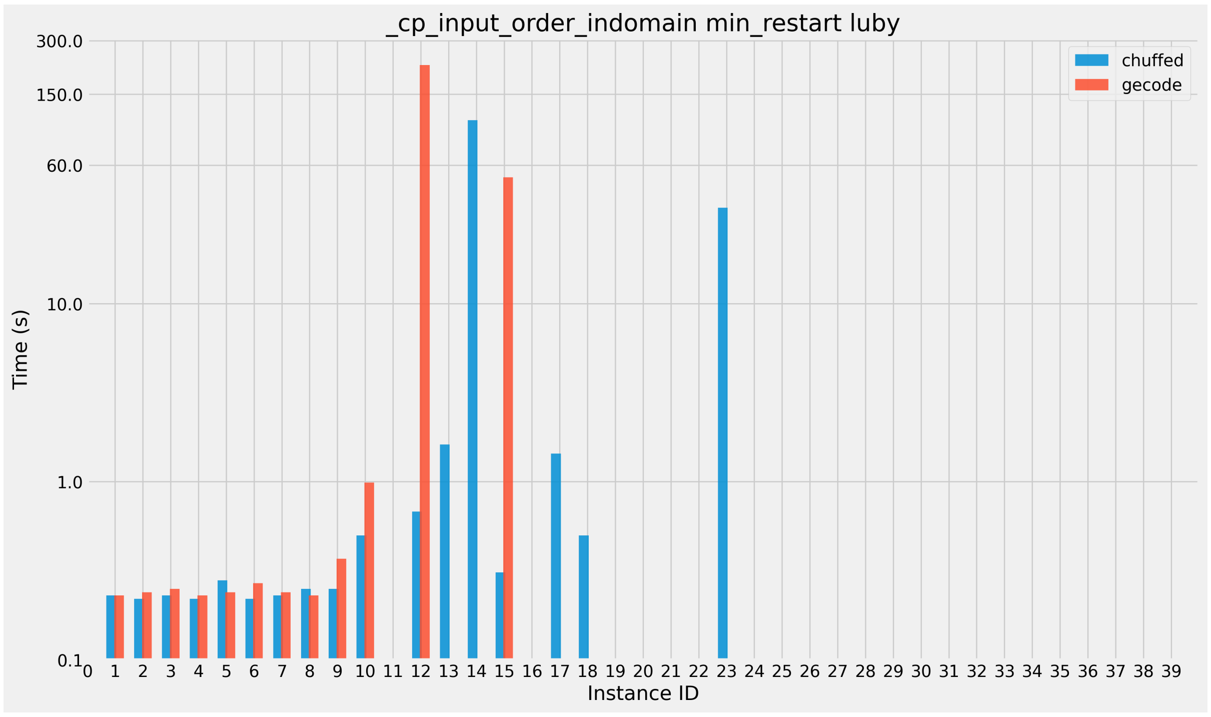

**Figure 5:** Time required to solve the instances for both solvers using input_order and indomain_min as annotations with a luby restart

| id | indomain min | indomain min + restart linear | indomain min + restart luby | indomain random |
|---|---|---|---|---|
| 1 | 12 | 12 | 12 | 12 |
| 2 | 9 | 9 | 9 | 9 |
| 3 | 10 | 10 | 10 | 10 |
| 4 | 11 | 11 | 11 | 11 |
| 5 | 12 | 12 | 12 | 12 |
| 6 | 13 | 13 | 13 | 13 |
| 7 | 14 | 14 | 14 | 14 |
| 8 | 15 | 15 | 15 | 15 |
| 9 | 16 | 16 | 16 | 16 |
| 10 | 17 | 17 | 17 | 17 |
| 11 | 18 | - | - | 18 |
| 12 | 19 | 19 | 19 | 19 |
| 13 | 20 | 20 | - | 20 |
| 14 | 21 | 21 | 21 | 21 |
| 15 | 22 | 22 | 22 | 22 |
| 17 | 24 | 24 | - | - |
| 18 | 25 | - | 25 | - |
| 23 | - | 30 | - | - |
| 28 | - | 35 | - | - |

**Table 5:** Results obtained only on the Gecode solver (since chuffed doesn't support the dom_wdeg annotation) using dom_wdeg as annotation and different restart strategies

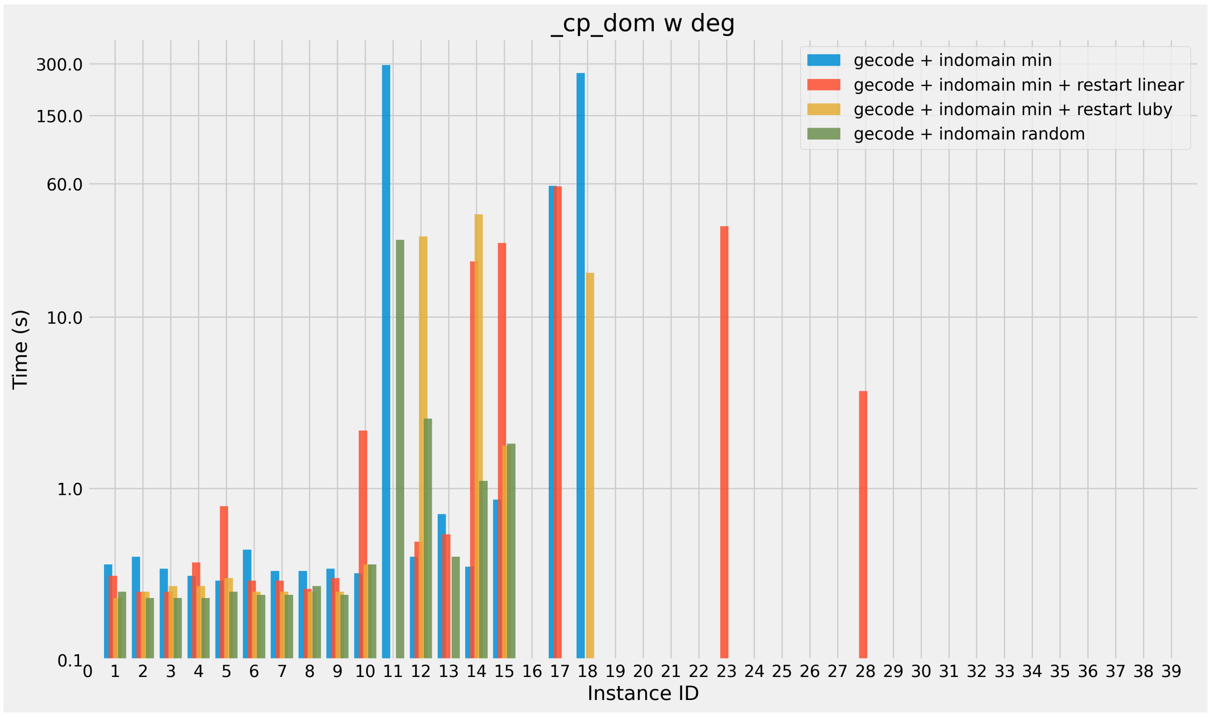

**Figure 6:** Time required to solve the instances for the Gecode solver using dom_wdeg as annotation and different restart strategies

### 2.6.3 Experimental Results for the second variant of the problem with rotation

In this section we report the results obtained for the second variant of the problem with rotation. Since this variant is more difficult to solve, the results are worse than the previous one. Also in this case applying the symmetry breaking constrain improves the results (1). Anyway using any search strategies and restart strategies worsen the results for both solvers (table 7, 8, 9 and 10).

| id | chuffed | chuffed + sb | gecode | gecode + sb |
|---|---|---|---|---|
| 1 | 9 | 9 | 9 | 9 |
| 2 | 9 | 9 | 9 | 9 |
| 3 | 10 | 10 | 10 | 10 |
| 4 | 11 | 11 | 11 | 11 |
| 5 | 12 | 12 | 12 | 12 |
| 6 | 13 | 13 | - | 13 |
| 7 | 14 | 14 | - | 14 |
| 8 | 15 | 15 | - | 15 |
| 9 | 16 | 16 | - | 16 |
| 10 | 17 | 17 | - | 17 |
| 12 | 19 | 19 | - | 19 |
| 13 | 20 | 20 | - | - |
| 14 | 21 | 21 | - | 21 |
| 15 | 22 | 22 | - | 22 |
| 18 | - | - | - | 25 |

**Table 6:** Results on both solvers with and without symmetry breaking constrain

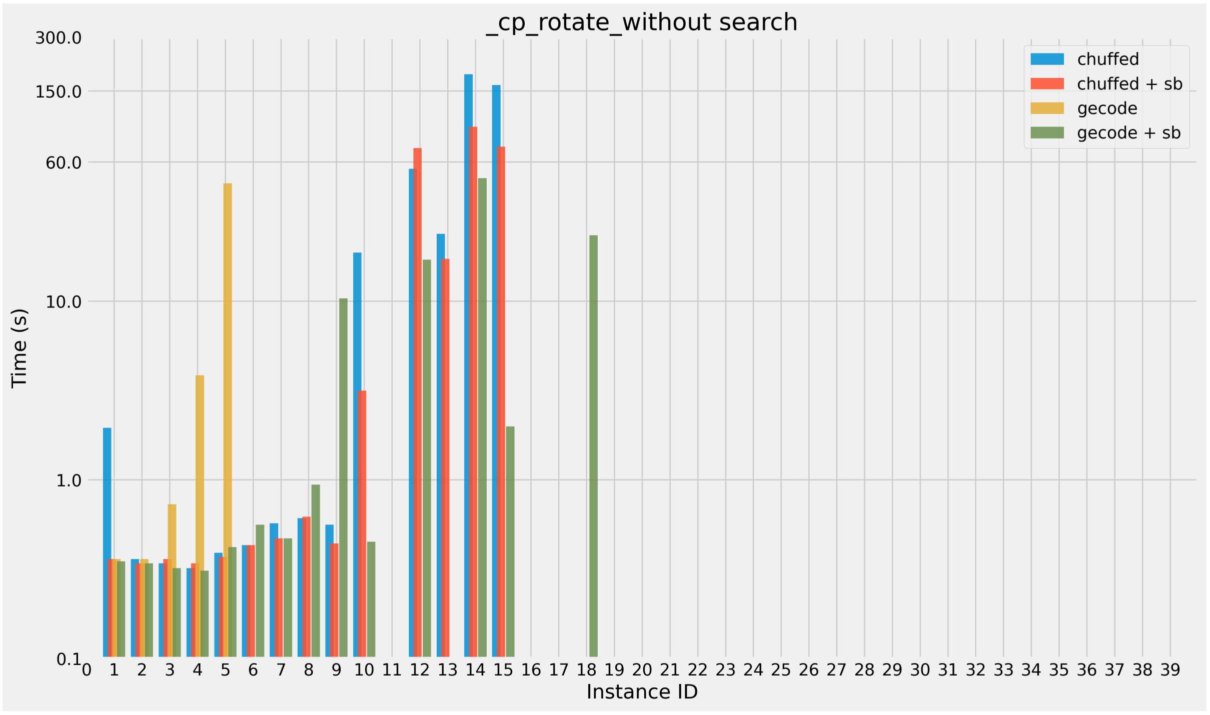

**Figure 7:** Time required to solve the instances for both solvers with and without symmetry breaking constrain

| id | chuffed | gecode |
|---|---|---|
| 1 | 9 | 12 |
| 2 | 9 | 9 |
| 3 | 10 | 10 |
| 4 | 11 | 11 |
| 5 | 12 | 12 |
| 6 | 13 | 13 |
| 7 | 14 | - |
| 8 | 15 | 15 |

**Table 7:** Results obtained on both solver with symmetry breaking constrain and using input_order and indomain_min as annotations

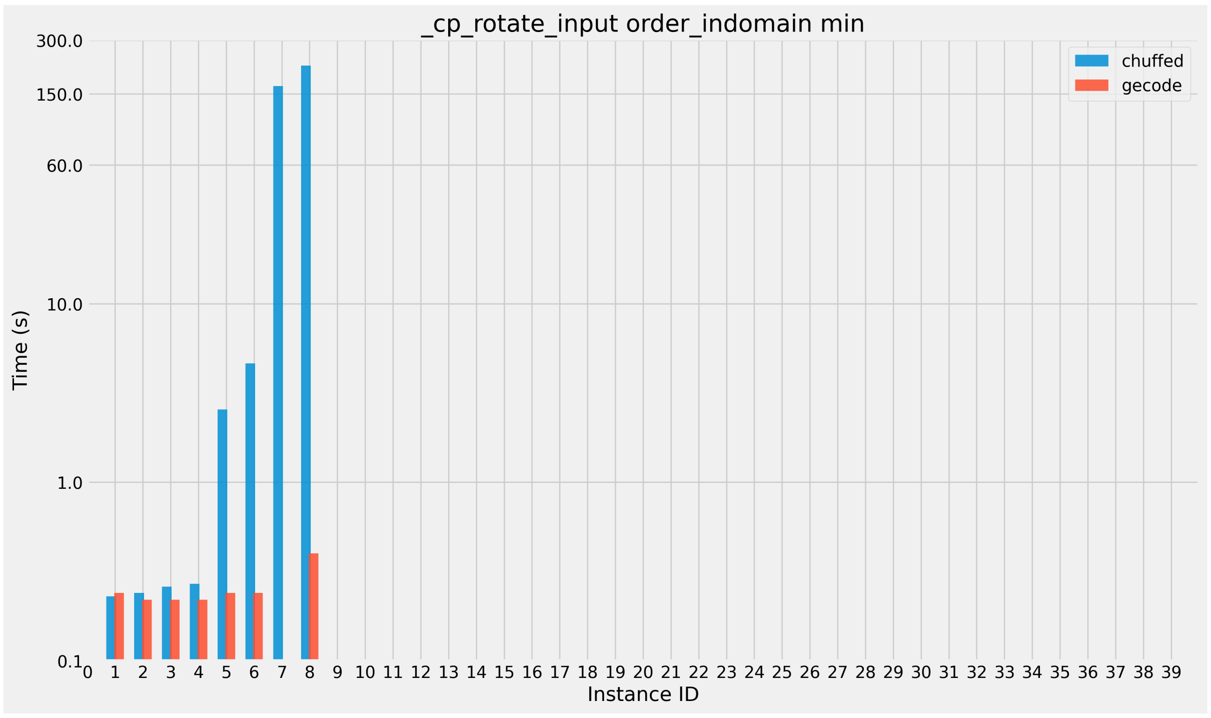

**Figure 8:** Time required to solve the instances for both solvers using input_order and indomain_min as annotations

| id | chuffed | gecode |
|---|---|---|
| 1 | 9 | 9 |
| 2 | 9 | 9 |
| 3 | 10 | 10 |
| 4 | 11 | 11 |
| 5 | 12 | 12 |
| 6 | 13 | 13 |
| 7 | 14 | - |
| 8 | 15 | - |

**Table 8:** Results obtained on both solver with symmetry breaking constrain and using input_order and indomain_min as annotations with a linear restart

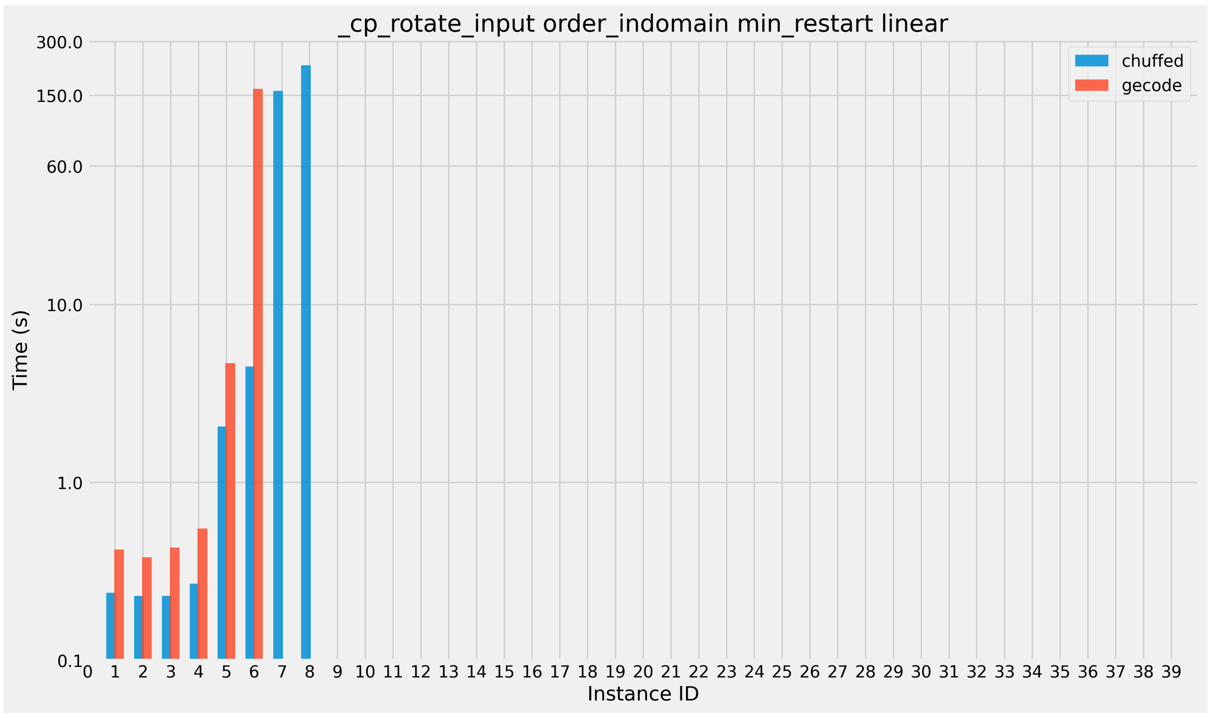

**Figure 9:** Time required to solve the instances for both solvers using input_order and indomain_min as annotations with a linear restart

| id | chuffed | gecode |
|---|---|---|
| 1 | 9 | 9 |
| 2 | 9 | 9 |
| 3 | 10 | 10 |
| 4 | 11 | 11 |
| 5 | 12 | 12 |
| 6 | 13 | 13 |
| 7 | 14 | - |

**Table 9:** Results obtained on both solver with symmetry breaking constrain and using input_order and indomain_min as annotations with a luby restart

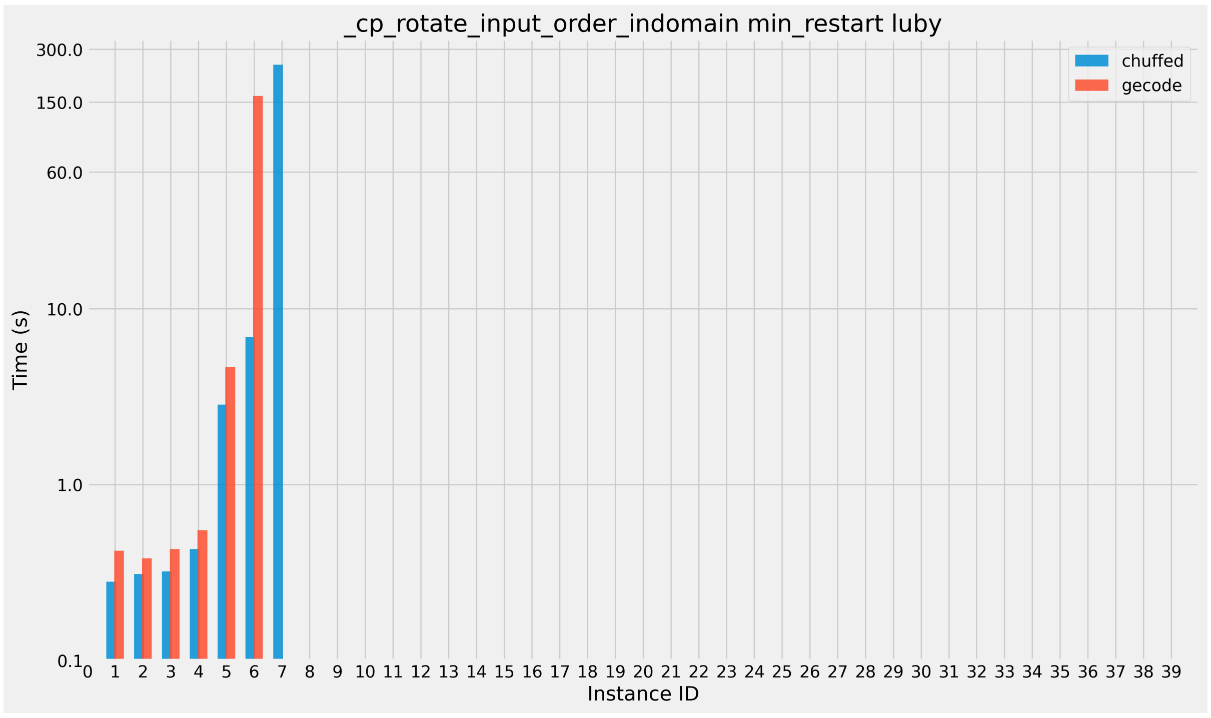

**Figure 10:** Time required to solve the instances for both solvers using input_order and indomain_min as annotations with a luby restart

| id | indomain min | indomain min + restart linear | indomain min + restart luby |
|---|---|---|---|
| 1 | 9 | 9 | 9 |
| 2 | 9 | 9 | 9 |
| 3 | 10 | 10 | 10 |
| 4 | 11 | 11 | 11 |
| 5 | 12 | 12 | 12 |
| 6 | - | 13 | 13 |
| 7 | 14 | - | - |
| 8 | 15 | - | 15 |
| 9 | 16 | - | - |

**Table 10:** Results obtained only on the Gecode solver (since chuffed doesn't support the dom_wdeg annotation) using dom_wdeg as annotation and different restart strategies

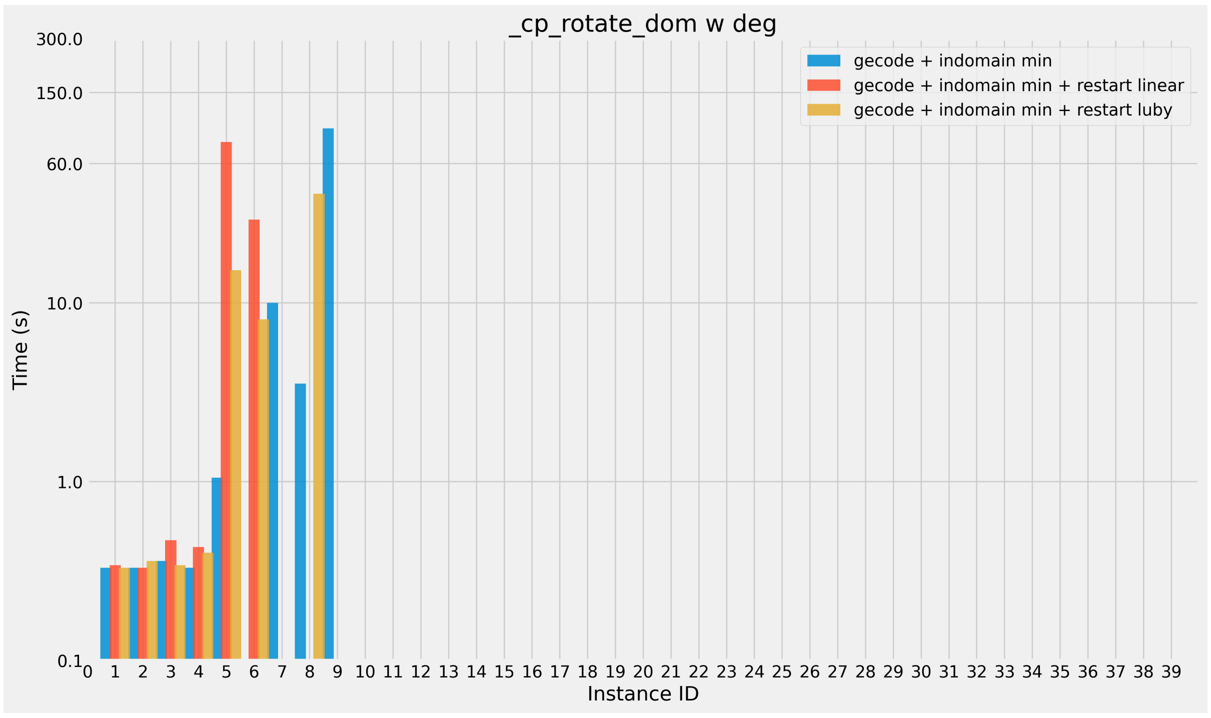

**Figure 11:** Time required to solve the instances for the Gecode solver using dom_wdeg as annotation and different restart strategies

## 3 SAT Model

The Boolean Satisfability problem (SAT) is a problem in which we are given a set of clauses. The goal is to find a truth assignment to the variables that makes all the clauses true. The formalization of the problem differ from CP and MIP. In particular, here we define an encoding that implicitly contain the length of the silicon plate and we try to find a solution that satisfy all the clauses for the smallest length $l$.

### 3.1 Decision variables

As described in the section 3, we encode the board as a 3d matrix of boolean variables. where each element is true if the circuit is present in that position as described in the equation 3.

$$board_{ijk} = \begin{cases} 1 & \text{if circuit is present in position (i,j,k)} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

### 3.2 Objective function

We want to minimize the length of the silicon plate, but what we are actually solving is if the length $l$ admit any solution. So in order to find the optimal solution we need to iteratively increase the length from the *l_min* to the *l_max* and check if the problem is satisfable. we defined *l_min* in the naive way, as length of the maximum circuit $\max_i \ y_i$, while *l_max* as the sum of all the length $y_i$ of the circuit.

$$l_{max} = \sum_{i=1}^{n} y_i \tag{4}$$

### 3.3 Constrains

The two main constrains we need to define are the non-overlapping of the circuits and the positioning of each circuit.

**Non-overlapping of the circuits** The idea is that each element of the plate can be occupied by at most one circuit. So we can define the constrain as the following:

$$\bigwedge_{0 \le i < w} \bigwedge_{0 \le j < l} \wedge \bigwedge_{0 \le k < n} \bigwedge_{0 \le p < n} \bigwedge_{k \ne p} \neg(board_{i,j,k} \wedge board_{i,j,p}) \tag{5}$$

**Positioning of the circuits** To position the circuits we need to define a possible positioning for each valid left bottom corner of each circuit. This means that given a valid corner ($\hat{x_i}$, $\hat{y_i}_i$) we need to set to true all the cell of the circuit (so from $\hat{x_i}$ to $x_i$ and from $\hat{y_i}$ to $y_i$) on the board. This is done by:

$$\text{exactly\_one\_he} = \text{at\_most\_one\_he(all\_position)} \wedge \text{at\_least\_one(all\_position)} \tag{6}$$

where *all_position* is a list of all the possible position of the circuit.

**Symmetry breaking** Given the symmetry identified in the previous section, we restrain the position of the highest circuit to be in the coordinate $(0, 0)$.

## 3.4 Rotation

Introducing the rotation of the circuits mean that we can position the circuit in more ways. While this does not affect the non-overlapping constraint, we need to add the new position to the positioning constrain. This is done by adding a new foreach to the algorithm described in in the positioning paragraph which invert the $.x$ and $.y$ of the circuit. Also to break the symmetry introduced if the circuit is a square, the rotation is only applied if the circuit is not a square.

## 3.5 Experiments

### 3.5.1 Experimental Design

**SETUP** The experiments are performed on a machine with the following specifications:

- Intel Core i5-8300H (4 core)
- Ram DDR4 8GB
- Nvidia GTX 1050 4GB
- Windows 10 Home x64
- z3-solver 4.12.1.0

**Design** To test the performance of the model we used the z3 library. we try to solve the problem with and without the symmetry breaking constrain.

### 3.5.2 Experimental Results

For the first variant of the problem, the results show that the symmetry breaking constrain help to improve the performance of the model (table 11) and also reduce the time required to solve each instance (figure 12). despite the second variant of the problem being more complex, the results are similar to the first variant (table 12).

| id | sat | sat + sb |
|---|---|---|
| 1 | 12 | 12 |
| 2 | 9 | 9 |
| 3 | 10 | 10 |
| 4 | 11 | 11 |
| 5 | 12 | 12 |
| 6 | 13 | 13 |
| 7 | 14 | 14 |
| 8 | 15 | 15 |
| 9 | 16 | 16 |
| 10 | 17 | 17 |
| 12 | 19 | 19 |
| 13 | - | 20 |
| 15 | - | 22 |

**Table 11:** results obtained on the first variant of the problem

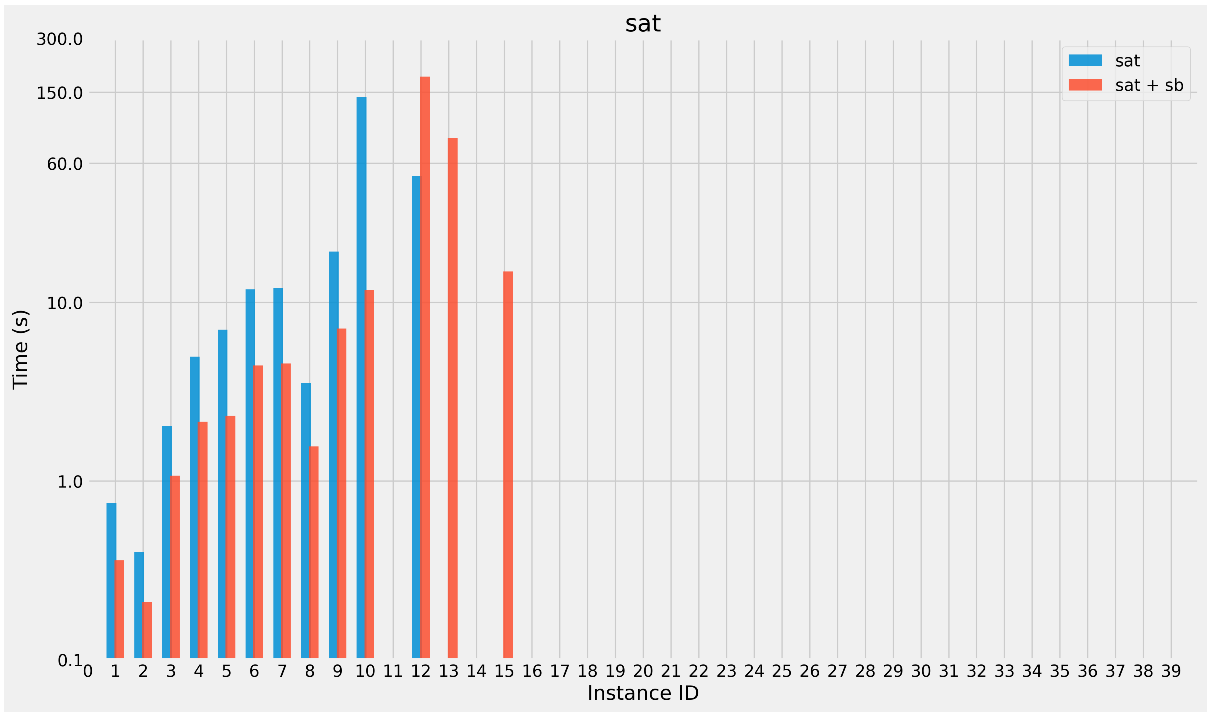

**Figure 12:** time require to solve each instance

| id | sat | sat + sb |
|---|---|---|
| 1 | 12 | 12 |
| 2 | 9 | 9 |
| 3 | 10 | 10 |
| 4 | 11 | 11 |
| 5 | 12 | 12 |
| 6 | 13 | 13 |
| 7 | 14 | 14 |
| 8 | 15 | 15 |
| 9 | 16 | 16 |
| 10 | 17 | 17 |
| 12 | 19 | 19 |
| 13 | 20 | 20 |
| 15 | - | 22 |

**Table 12:** results obtained on the second variant of the problem with rotation

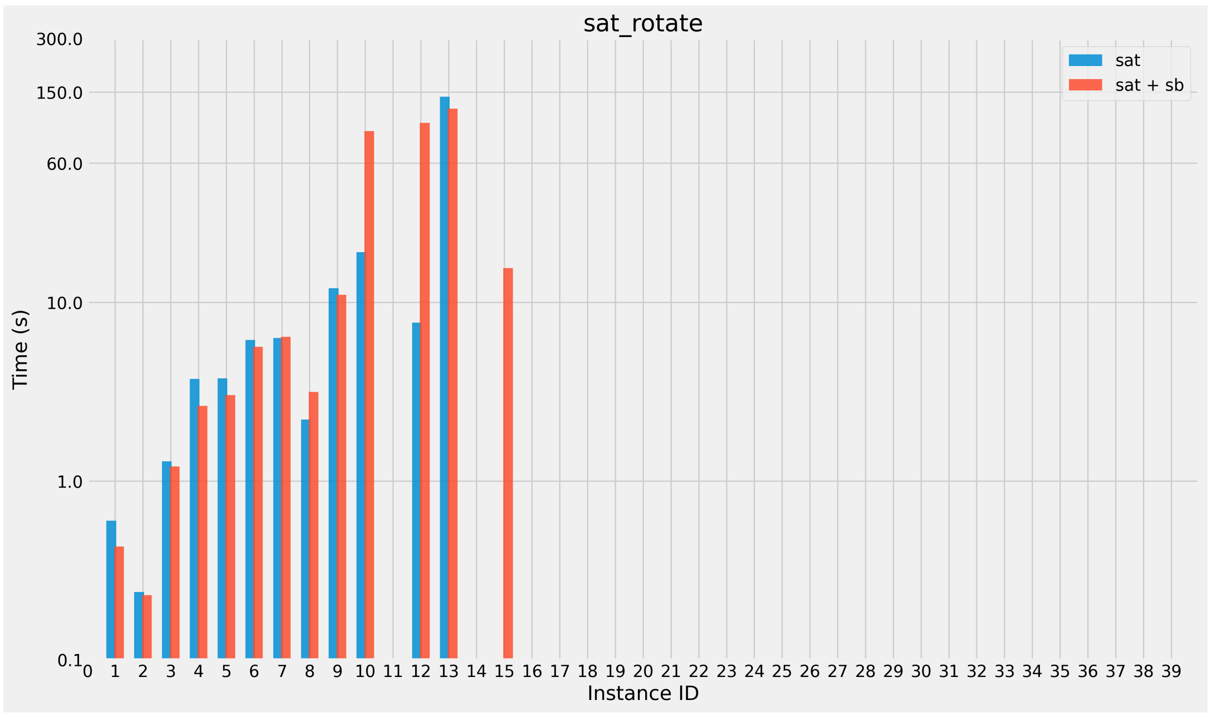

**Figure 13:** time required to solve each instance

## 4 MIP Model

Mixed Integer Programming (MIP) is a class of optimization problems that can be formulated as a linear program (LP) with integer variables. The goal of MIP is to find the optimal solution to a problem subjec to constraints, where some of the variable are restricted to be integers.

### 4.1 Decision variables

The decision variables are the same as the CP model 2.1 and are defined as follows:

- $\hat{x}$: with domain $[0,\ w]$, array of size *CIRCUITS*, where each element is the bottom left corner of the circuit in the x-axis.
- $\hat{y}$: with domain $[0,\ l\_max]$, array of size *CIRCUITS*, where each element is the bottom left corner of the circuit in the y-axis.
- $l$: with domain $[0,\ l\_max]$, is the length of the silicon plate.

### 4.2 Objective function

The objective function is the same as the CP model 2.2, which consists in minimizing the length of the silicon plate.

### 4.3 Constrains

The constrains are the same as the CP model 2.3 with the exception of the non-overlapping constrain, which is different in the MIP model.

**Non-overlapping constrain** Following Kalvelagen 2017, Considering two circuits $i$ and $j$, there are basically four cases to consider:

- $\hat{x_i} + x_i \leq \hat{x_j}$
- $\hat{x_j} + x_j \leq \hat{x_i}$
- $\hat{y_i} + y_i \leq \hat{y_j}$
- $\hat{y_j} + y_j \leq \hat{y_i}$

From those cases, we can formulate the non-overlapping constrain as follows:

$$\begin{aligned}
\hat{x_i} + x_i \leq \hat{x_j} + \delta_{i,j,1} \times w \quad & \text{or} \\
\hat{x_j} + x_j \leq \hat{x_i} + \delta_{j,i,1} \times w \quad & \text{or} \\
\hat{y_i} + y_i \leq \hat{y_j} + \delta_{i,j,2} \times l\_max \quad & \text{or} \\
\hat{y_j} + y_j \leq \hat{y_i} + \delta_{j,i,2} \times l\_max \quad & \text{or} \\
\delta_{i,j,1} + \delta_{i,j,2} + \delta_{j,i,1} + \delta_{j,i,2} \leq 3 &
\end{aligned}$$

### 4.4 Rotation

To model the rotation of the circuits, following the approach of 2.4, we need to add a new variable $rotation_i$, which is a binary variable that indicates if the circuit $i$ is rotated or not. Then the actual width and height of the circuit $i$ are managed by two variables $x\_r_i$ and $y\_r_i$.

**Rotation constrain** The same constrain for the rotation defined in the CP model is applied here.

## 4.5 Experiments

### 4.5.1 Experimental Design

**SETUP** The experiments are performed on a machine with the following specifications:

- Intel Core i5-8300H (4 core)
- Ram DDR4 8GB
- Nvidia GTX 1050 4GB
- Windows 10 Home x64
- pulp 2.7.0

**Design** In this case we compare the performance of the MIP model using two solver. The first one is the CBC, which is an open source solver included in the pulp library. The second one is the Gurobi Gurobi Optimization, LLC 2023, which is a commercial solver that come with a free academic license. Furthermore, we compare the performance of the MIP model with and without the symmetry breaking constrain.

### 4.5.2 Experimental Results

The tables show that the Gurobi solver give the best results and the symmetry breaking constrain improve the results (table 14 and figure 14). Since the results obtained using the CBC solver are not satisfying, we will only use the Gurobi solver for the second variant of the problem. In the second variant of the problem, the symmetry breaking constrain worsen the results (table 15 and figure 15).

**Table 13:** Results for the MIP model using Gurobi and the CBC solver.

| id | cbc | cbc + sb | gurobi | gurobi + sb |
|---|---|---|---|---|
| 1 | 12 | 12 | 12 | 12 |
| 2 | 9 | 9 | 9 | 9 |
| 3 | 10 | 10 | 10 | 10 |
| 4 | 11 | - | 11 | 11 |
| 5 | 12 | - | 12 | 12 |
| 6 | 13 | - | 13 | 13 |
| 7 | 14 | - | 14 | 14 |
| 8 | 15 | - | 15 | 15 |
| 9 | 16 | - | 16 | 16 |
| 10 | 17 | - | 17 | 17 |
| 12 | - | - | 19 | 19 |
| 13 | - | - | 20 | 20 |
| 14 | - | - | - | 21 |
| 15 | - | - | 22 | 22 |
| 17 | - | - | 24 | 24 |
| 18 | - | - | 25 | 25 |
| 23 | - | - | 30 | 30 |
| 24 | - | - | 31 | 31 |
| 27 | - | - | 34 | 34 |
| 28 | - | - | 35 | 35 |
| 29 | - | - | 36 | 36 |
| 31 | - | - | - | 38 |

**Table 14:** Results for the first variant of the problem

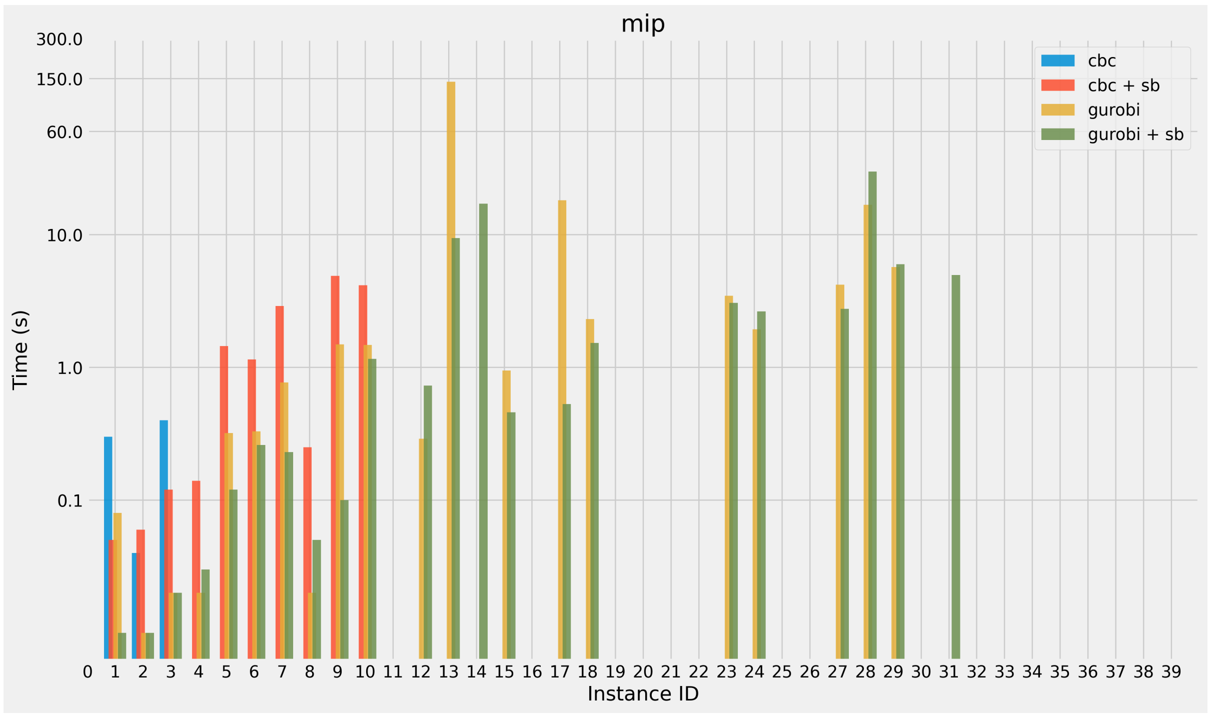

**Figure 14:** Time required to solve each instance for each solver.

| id | gurobi | gurobi + sb |
|---|---|---|
| 1 | 12 | 12 |
| 2 | 9 | 9 |
| 3 | 10 | 10 |
| 4 | 11 | 11 |
| 5 | 12 | 12 |
| 6 | 13 | 13 |
| 7 | 14 | 14 |
| 8 | 15 | 15 |
| 9 | 16 | 16 |
| 10 | - | 17 |
| 12 | 19 | 19 |
| 15 | 22 | 22 |
| 17 | 24 | 24 |
| 18 | 25 | - |
| 23 | - | 30 |
| 24 | 31 | 31 |
| 26 | 33 | - |
| 27 | 34 | 34 |
| 28 | 35 | - |
| 29 | - | 36 |
| 31 | 38 | - |

**Table 15:** Results for the second variant of the problem

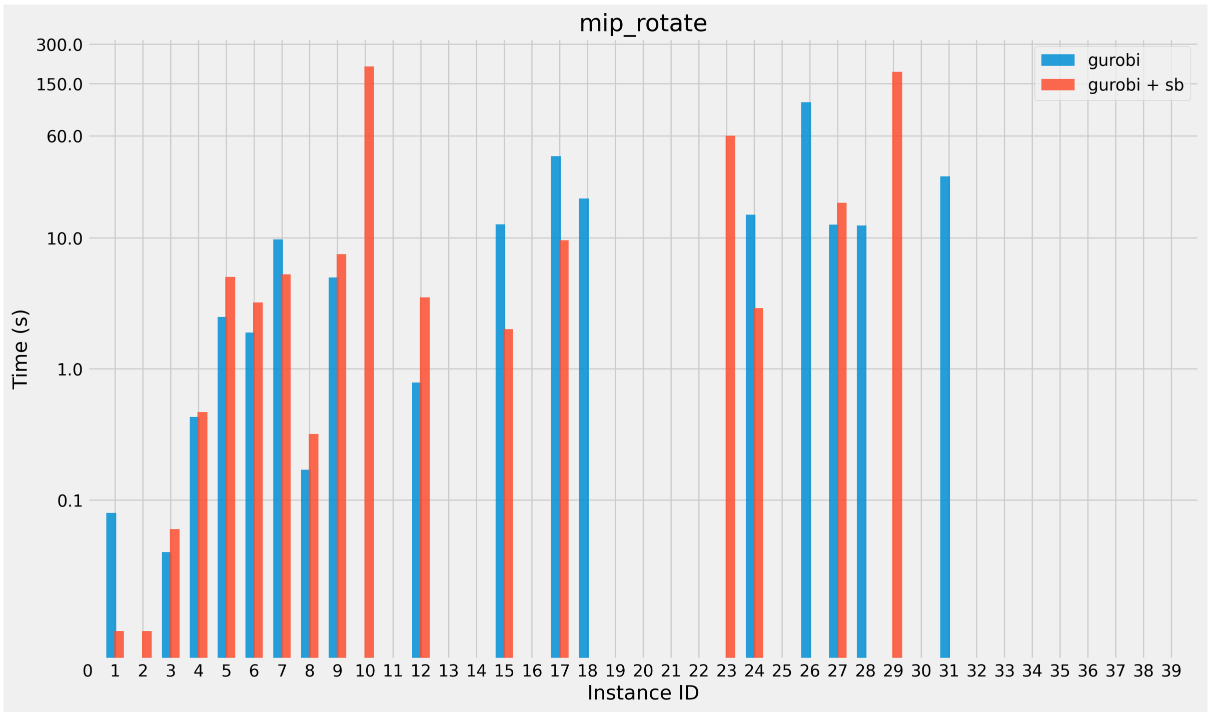

**Figure 15:** Time required to solve each instance

## 5 Conclusion

From the table 16 we can see that we find the most solution using MIP with the Gurobi solver and CP using the chuffed solver. However, the Gurobi solver is slightly faster than the chuffed one with an average of *3.02* seconds per instance against *6.01*. For the second variant of the problem, the chuffed solver find less number of solution, while the other solvers find the same number. In this case Gurobi is the best. However, since the search space increase also the time required for finding the optimal solution increase bringing the average time required to *31,54*.

| id | mip | mip_rotate | sat | sat_rotate | cp | cp_rotate |
|---|---|---|---|---|---|---|
| 1 | 12/0.01 | 12/0.01 | 12/0.36 | 12/0.43 | 12/0.22 | 9/0.35 |
| 2 | 9/0.01 | 9/0.0 | 9/0.21 | 9/0.23 | 9/0.22 | 9/0.34 |
| 3 | 10/0.02 | 10/0.04 | 10/1.07 | 10/1.21 | 10/0.22 | 10/0.32 |
| 4 | 11/0.02 | 11/0.43 | 11/2.15 | 11/2.64 | 11/0.22 | 11/0.31 |
| 5 | 12/0.12 | 12/2.5 | 12/2.32 | 12/3.03 | 12/0.25 | 12/0.37 |
| 6 | 13/0.26 | 13/1.9 | 13/4.44 | 13/5.65 | 13/0.23 | 13/0.43 |
| 7 | 14/0.23 | 14/5.28 | 14/4.55 | 14/6.32 | 14/0.23 | 14/0.47 |
| 8 | 15/0.02 | 15/0.17 | 15/1.56 | 15/2.21 | 15/0.25 | 15/0.61 |
| 9 | 16/0.1 | 16/4.98 | 16/7.14 | 16/11.02 | 16/0.26 | 16/0.44 |
| 10 | 17/1.16 | 17/203.16 | 17/11.69 | 17/19.09 | 17/0.31 | 17/0.45 |
| 11 | - | - | - | - | 18/30.85 | - |
| 12 | 19/0.29 | 19/0.79 | 19/51.03 | 19/7.71 | 19/0.48 | 19/17.06 |
| 13 | 20/9.46 | - | 20/82.87 | 20/121.2 | 20/0.34 | 20/17.25 |
| 14 | 21/17.11 | - | - | - | 21.0/0.86 | 21.0/48.85 |
| 15 | 22/0.46 | 22/2.01 | 22/14.88 | 22/15.54 | 22/0.31 | 22/1.99 |
| 16 | - | - | - | - | - | - |
| 17 | 24/0.53 | 24/9.62 | - | - | 24/1.21 | - |
| 18 | 25/1.53 | 25/19.94 | - | - | 25/0.49 | 25/23.33 |
| 20 | - | - | - | - | - | - |
| 23 | 30/3.07 | 30.0/60.27 | - | - | 30/35.72 | - |
| 24 | 31/1.94 | 31.0/2.91 | - | - | - | - |
| 26 | - | 33/108.53 | - | - | 33/8.28 | - |
| 27 | 34/2.76 | 34/12.64 | - | - | 34/34.67 | - |
| 28 | 35/16.79 | 35.0/12.42 | - | - | 35/13.04 | - |
| 29 | 36/5.72 | 36.0/185.16 | - | - | - | - |
| 31 | 38/4.97 | 38/29.49 | - | - | 38/3.71 | - |
| 33 | - | - | - | - | - | - |
| 36 | - | - | - | - | 22/6.01 | - |

**Table 16:** Number of instances solved by each solver and the time required in seconds.

## References

Gurobi Optimization, LLC (2023). *Gurobi Optimizer Reference Manual.* URL: https://www.gurobi.com.

Kalvelagen, Erwin (July 11, 2017). *Rectangles: no-overlap constraints.* Rectangles: no-overlap constraints.

Stuckey, Peter J, Kim Marriott, and Guido Tack (n.d.). “MiniZinc Handbook”. In: ().