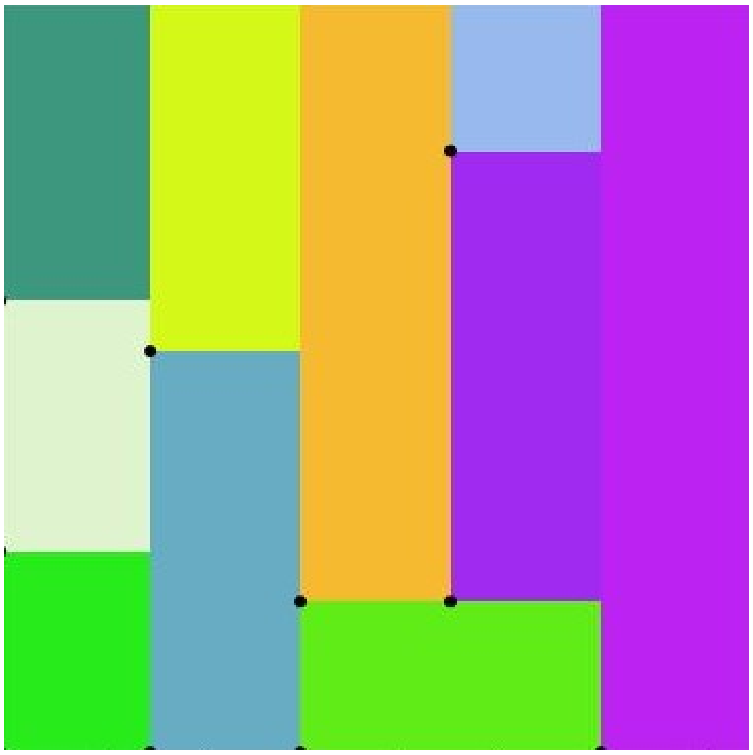
VLSI Project

Valentina Boriano and Simone Vizzuso
valentina.boriano@studio.unibo.it and simone.vizzuso@studio.unibo.it
Code: https://github.com/valentinaboriano/VLSI-Project

February 1, 2023



1 Introduction

VLSI is an optimization problem with a plate having fixed width and virtually infinite height. Taking in input n circuits, an optimizer need to fit them in less space available with the minimum possible height. So the common parameters between the three model will be the **plate width** that is fixed for every instance, as the **number of circuits** and their **heights** and **widths**. Those are integer numbers and cannot be changed or altered (the only exception is in the rotation model where it's possible to swap height and width but the two values must remain the same).

Those values are taken as input from 40 instances files which also function as a schema for the output. The output have the values in the same location but will also add the coordinates next to the heights of the relative circuits and a value indicating whether or not in the relative solution that particular circuit has been rotated.

In this report we will explain how we modeled the VLSI Problem in CP, SMT and MIP. All the solutions images and texts, that are in GitHub code, are made by our best models of CP, SMT and MIP. But you can find all the statistics for all our solver and search strategies tested. We divided our project in **CP, SMT** and **MIP** folders. In each, you can find the following directories:

```
🟥 CP/SMT/MIP
│ ├── 🟧 mzn (only in CP)
│ ├── 🟧 py
│ └── 🟧 solution
│       ├── 🟩 model
│       └── 🟩 rotation
├── 🟥 Instances
└── 🟥 Plot
```

In both, model and rotation folder you can find:

```
🟩 model/rotation
  ├── 🟪 images
  ├── 🟪 statistics
  │     └── 🟪 plot
  └── 🟪 txt
```

The **instances** folder contains all the instances. **Plot**, is composed by plot.py that has a method to make the bar plots. In **mzn** and **py** folders, you can find all the files with that extension. Solution, contains:

- All the **images** of the plates with circuits for each instance
- All the **statistics** with all the solvers, search strategies, free search and symmetry breaking tested for each instance and the **plot**s comparing their height and time.
- All the **texts** composed by the width and height of the plate, number of circuits and solution for each instance with our best models

Between the three models, the majority of inputs, variable and the objective function are in common. There are a few differences in a pair of them that will be explained in the relative model section.

As said before, the input parameter are represented such as: number of circuit as **n_circuit**, plate width with **plate**, heights and widths of the circuits respectively **y** and **x**. n_circuits and plate are simple integer value while x and y are two integer lists that contain in the i^{th} position the width and height of the i^{th} circuit.

Two variables that are in common to each model are the arrays with length [1...n_circuits]: ***coord_x*** contains all x coordinates of circuits and ***coord_y*** contains all y coordinates of circuits. Their domain is respectively [0...***plate***-min($\boldsymbol{x}$)] and [0...***height_max***-min($\boldsymbol{y}$)]. We decided to not just have as upper bound

plate or ***height*** because we have thought that the coordinates can't reach a value bigger than the difference between the ***plate*** or the ***height*** and the minimum of $\boldsymbol{x}$ or the minimum of $\boldsymbol{y}$. Following this approach allow us to reduce the search space for both variables. The variable ***coord_x*** means that in ***coord_x***$_i$ there is the x coordinate of the i^{th} circuit. The same is for ***coord_y***.

The other two value are **height_min** and **height_max**, that also represent the lower and upper bound of the objective variable **height**, are defined as follows:

- **height_min** is computed by the maximum value between the plate and the maximum value from the **y** (heights). This choice is determined by the observation that the height found in the solution could not be physically lower than the highest circuit. On the other hand, when there are a lot of small circuits it is better to choose a higher value to optimize the search strategy.
- **height_max** is determined in two different ways: if half of the total sum between the heights are lower than the highest circuit the value will be the sum between all the heights. If not, the value will be half of the total sum between the heights. These choices are realized because if there is a circuit much bigger than all the other one, half of the total sum could be lower.

For what concern the rotation model it has been used three variable. The first is **rotation**, a boolean list that point out if the i^{th} circuit in the solution has been rotated or not. A rotated circuit is used with width and height swapped. The other two variable for CP and SMT are **rot_x** and **rot_y** that are used when a circuit is rotated. They took the value originally contained in y and x respectively and are treated as the width and the height in the rotation solution. Another change is about the bounds of coord_x and coord_y. In the rotation case, the bound take [0...***plate***-min(min($\boldsymbol{x}$), min($\boldsymbol{y}$))] and [0...***height_max***-min(min($\boldsymbol{x}$), min($\boldsymbol{y}$))]. This decision was taken because if a circuit is rotated, its heights and its width will change and so the circuit will not fit in the plate without changing the boundaries.

The objective variable is **height** and is an integer value. It's bound are, as said, height_min for the lower bound and height_max for the upper bound. The objective is to minimize this value.

We have decided to divide the project in two, the student Boriano focused more on CP model, Vizzuso on SMT, and both on MIP. This project took us 4 weeks to complete it properly. To have all the statistics with different solvers, we utilized some days. We had some difficulties to work with the minizinc and PuLP libraries, due to the lack of documentation, we had to search and try many times to have a working model. Another solver in SMT that we would have liked to tried is cvc5 but we have not been able to use it on windows and we couldn't use Ubuntu.

2 CP Model

2.1 Decision Variables

For our CP model, we needed 3 different decision variable: ***height***, ***coord_x*** and ***coord_y*** that are already marked out in Section 1.

2.2 Objective function

For our CP model, we minimize the height because our aim is to have all circuits in the smallest space. The bounds of the height are already described on Section 1.

2.3 Constraints

We impose 4 constraints. The first problem that came to us in mind is that the circuits must stay in the plate area, so we added ***forall(i in 1..n_circuits)***$(coord\_x[i] + x[i] \leq plate \wedge coord\_y[i] + y[i] \leq height)$.

Then we have seen that all the circuits were overlapping. To solve this problem we use the global constraint ***diffn(coord_x, coord_y, x, y)*** which constrains i^{th} circuits, given by their origins (coord_x[i], coord_y[i]) and sizes (x[i], y[i]) to be non-overlapping.

Once solved the overlapping problem, we have seen that between circuits there was space, so we needed another constraint. To avoid this problem we impose two new constraints with the global constraint ***cumulative(coord_x, x, y, height)*** and ***cumulative(coord_y, y, x, plate)***.

Symmetry breaking constraints We run the model plotting all the solutions that it could find. We have seen that the mostly symmetries were about the axes. Then, once imposed these constraints, we have seen that there was another type of symmetry that the model swapped the circuits with the same shape.

So, we imposed 3 Symmetry Breaking constraints:

- To break the symmetry of the axis x
- To break the symmetry of the axis y
- To break the symmetry of the changing position of the circuits with the same shape

2.4 Rotation

For the rotation model, we add three variables ***rotation***, ***rot_x*** and ***rot_y*** already described in Section 1.

So now, we needed to impose new constraints on ***rotation*** variable, in which we enforce the False value for all that circuits that:

- Has its height the same as its width
- If rotated, then the new size is bigger than ***plate*** or ***height*** (E.g. A circuit has shape (2,15) and the plate width is 8, it cannot be rotated because it would be larger than the plate.

All the other constraints are the same of no rotation model that are already described in Section 2.3.

2.5 Validation

Experimental design To do this project, we decided to utilize Pycharm. On the terminal we run ***compiler.py*** which complete all the instances with our best model with and without rotation. To run it, firstly you need to change the path of the project. On src/CP/py directory, you can find compiler.py. In the compiler method there's a string called ***PATH*** which you can modify. Then, on the terminal, you need to go to the directory in which you downloaded Minizinc and then run the compiler.py:

```
C:\Program Files\MiniZinc> python PATH/src/CP/py/compiler.py
```

If you want to change :

- **Solver**. On src/CP/py directory, you can find compiler.py. In the compiler method there's a string called ***solver*** in which you can write which solver you want to try.
- **Free Search**. On src/CP/py directory, you can find compiler.py. In the compiler method (in the end) there's a boolean called ***free_search*** in which you can decide if you want to run with or without free search.
- **Instance** On src/CP/py directory, you can find compiler.py. In the compiler method (in the end) there are two integers ***from_instance*** and ***to_instance*** where you can write from which instance you want to start and the last instance you want to test
- **Search Strategy**. On src/CP/mzn directory, you can find final_model.mzn or rotation_model.mzn. You can try all the search strategy you need.
- **Symmetry Breaking**. On src/CP/mzn directory, you can find final_model.mzn or rotation_model.mzn. You can delete or keep the symmetry breaking.

We implemented the model utilizing two solvers: **Gecode** and **Chuffed**.

For both solvers, we tested different search strategies, we saved the results in CSVs files and then we plotted a graph to compare them for the **time utilized** to find the solution and for the **height found**.

For all these tests, we kept the free search because it improves a lot the quality of the results. In this report, we are going to show the most significant tests we made. In fact, many plots of the heights and some of the time won't be shown because the search strategies utilized, have found the same solution in approximately the same time. If you want to consult them, you can find it on the project (on the directory src/CP/model/statistics/plot). All the names of plots and statistics are given in the same way.

Plots if there is one solver:

```
solver_freesearch_True/False_restart1_value/variable-choices1-restart2_value/variable-
choices2.jpg
```

Plots if there are both solvers:

```
solver1_freesearch_True/False_restart_value/variable-choices-solver21_freesearch_True/
False_restart_value/variable-choices.jpg
```

Statistics:

```
solver_freesearch_True/False_restart_value/variables-choices.csv
```

Firstly, to test the restart, we imposed the following search strategy for all. We decided to put **input_order** and **indomain_min** for the variable ***height*** because this can enforce the minimum value. And **first_fail** for ***coord_x*** and ***coord_y*** in order that it can try different coordinates without following an order, and **indomain_min** so that all the circuits have not much space between them.

Once established the above variables, we tested the restart with **luby**, **constant** and **linear**. We start showing the results, for the model without rotation, for both solvers.

For **luby** we utilized 50, 100 and 200. For chuffed solver the time utilized for all the instances was more optimized with luby 50. The solution found were all the same. For gecode the search strategies are mostly the same, but we decided to keep luby 100, because in some instances used less time.

Already after our first test is evident the difference between the two solvers. Chuffed is much faster and allows us to solve all the instances, in contrast to gecode.

Then we tested restart **constant** and **linear** with the values 200 and 500. For both values, type of restart and solvers, the time taken to solve each instance and the solutions found are approximately the same.

We decided to keep constant 200 and linear 500 because they use less time for both solvers.

After, we took the best restart variable and value for both solver and we compared them.

For the Gecode the best restart are: **linear 500**, **constant 200** and **luby 100**. The time taken to solve the instances is the same for each restart variable, with a little improve on constant 200.

In the figure below, we have the best restart for chuffed: **linear 500**, **constant 200** and **luby 200**. You can see that luby 200 takes much less time in mostly each instance.

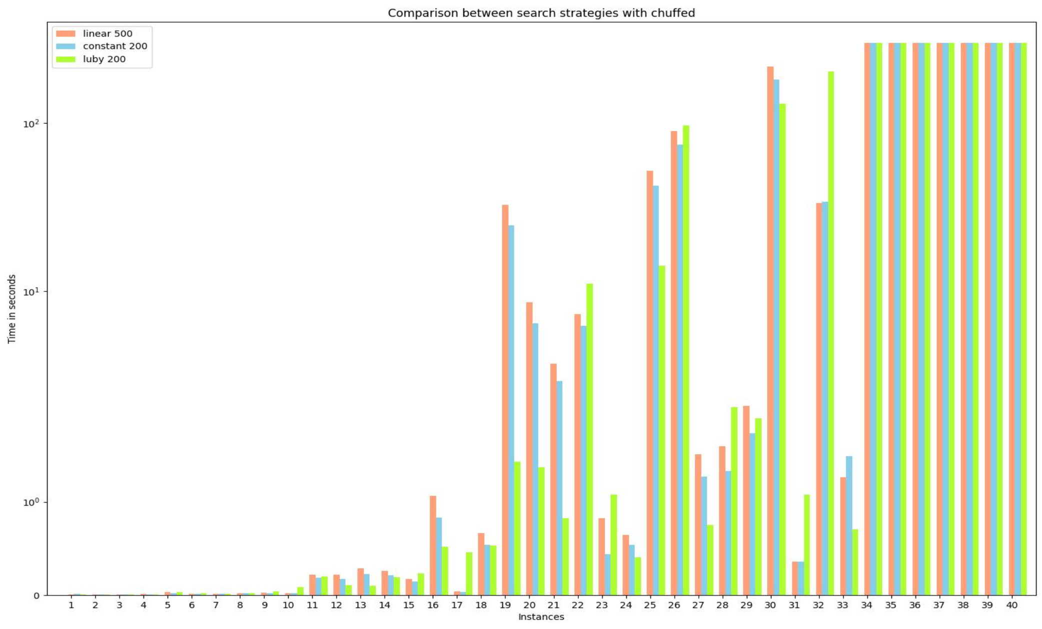


Figure 1: Comparison between linear 500, constant 200 and luby 200 with Chuffed solver

Then we tested different variables choice annotation. We focused mostly on ***input_order*** and ***first_fail***. We computed our model with our best restart variable, changing the variable choice annotation. We tested it with all the variables (height, coord_x and coord_y) with ***first_fail*** (indicated on the files as FF) and all with ***input_order*** (indicated on the files as IO). Comparing them with the variables choices annotation chosen at the beginning (indicated on the files as both).

For chuffed the variable first_fail take too much time. The best choice for this model is all **input_order**.

For gecode, the choice of the variable annotation change a lot the results for each instance. In fact, not only the time changes but also the instances that the solver is able to solve. The solution with all first_fail and the one in the beginning have the same behavior in time and height found. In the other hand, all input_order is able to find the solution on more complex instances but not on the easier.

The best solver is chuffed, in fact over speed also the results are better, respect to gecode. Another improving, was given by the use of free search, in fact it allows our model to obtain more optimal results. We also tested the model without symmetry breaking with and without free search to see how symmetry breaking improve our model. To compare them, we decided, not only to show the height but also the time utilized to find the optimal or a satisfying solution.

The first plot shows the comparison of time usage between chuffed and gecode solver utilizing free search and symmetry breaking. It's obvious how chuffed is faster and works better.

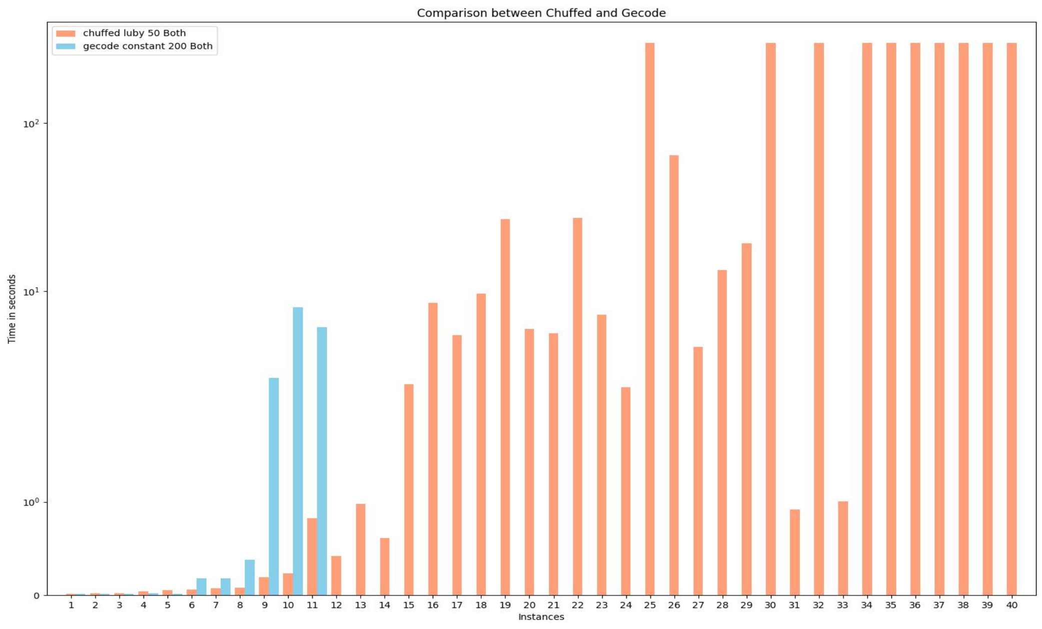


Figure 2: Comparison between our best models with chuffed and gecode on time

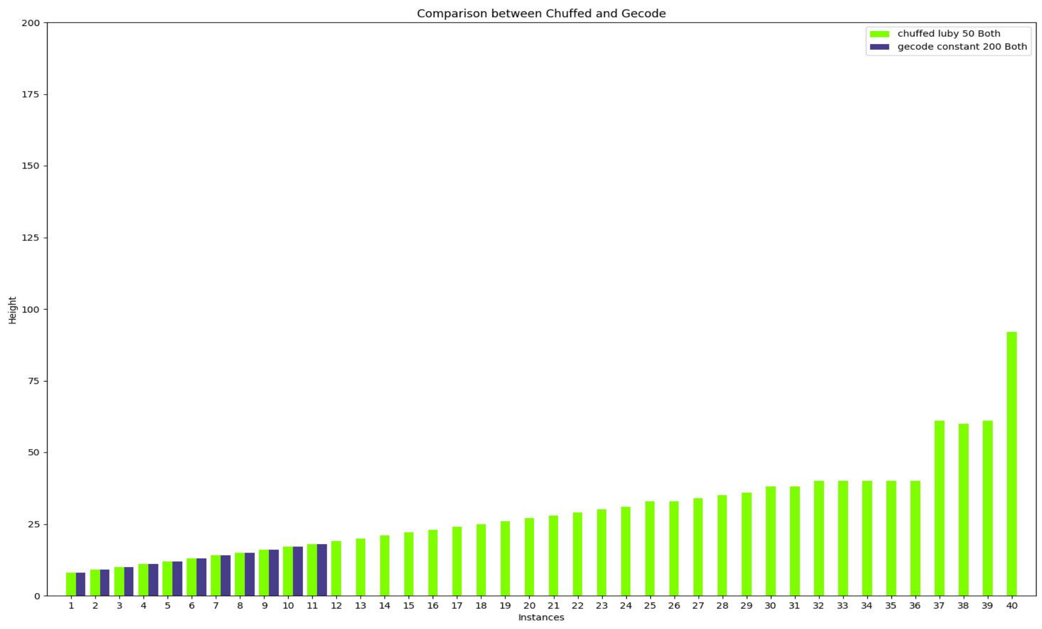


Figure 3: Comparison between our best models with chuffed and gecode on height

Then we did the same procedure for rotation model on both solvers, starting with **luby** with values 50

100 and 200.

For chuffed, luby 200 is the more unsuitable because in the instances n. 21 and 22 the height is superior to the others. And, for the instance n.38, only luby 50 is able to find a better solution. Also in terms of time, luby 50, is more convenient than the others. For gecode, solutions and time utilized with luby is mostly the same. Just luby 50 take less time respect to the others.

We also tested restart constant and linear with values 200 and 500. For both solvers, the solutions found and the time taken is similar. A small improvement is on constant 200 and linear 500 for both solvers.

Then we compared all the bests results for each solver: **luby 50**, **constant 200** and **linear 500**. Also in this case, all the best restart does not produce significant improvements. For Chuffed it is a bit better **luby 50** and for gecode **constant 200**.

Once found the best restart, like for the model without rotation, we tested the variable annotation in the same way as described above. For chuffed, the search strategy that make the solution worse is all input_order. In fact, for the instance 22 the height is bigger and also the time taken is much higher. We decided to keep both because on the instance 38 the solution is better, and also the time utilized is less. For gecode, all the search strategies solve until the instance 11, just all first fail and both can solve the 12^{th}. We kept both due to less time utilized.

In the end, we tested chuffed and gecode without symmetry breaking and free search to check its support.

Our hardware set up, to do all these tests, is composed by:

- **CPU** Intel(R) Core(TM) i7-8550U
- **RAM** 8 GB

Experimental results Now, we tested the model without symmetry breaking with and without free search to see how they improve our model. For all the following results, we utilized our best search strategies and the time limit of **5 minutes**. Firstly we show the time taken, it follows the height.

As you can see in figure 4 and in figure 5 the free search is fundamental to improve the time. In fact, the worst results are on the test made without free search. For symmetry breaking on chuffed, seems to not have any influence on the model, but is more evident on gecode when the time taken to solve an instance is higher, we can see how symmetry breaking helps to reduce time.

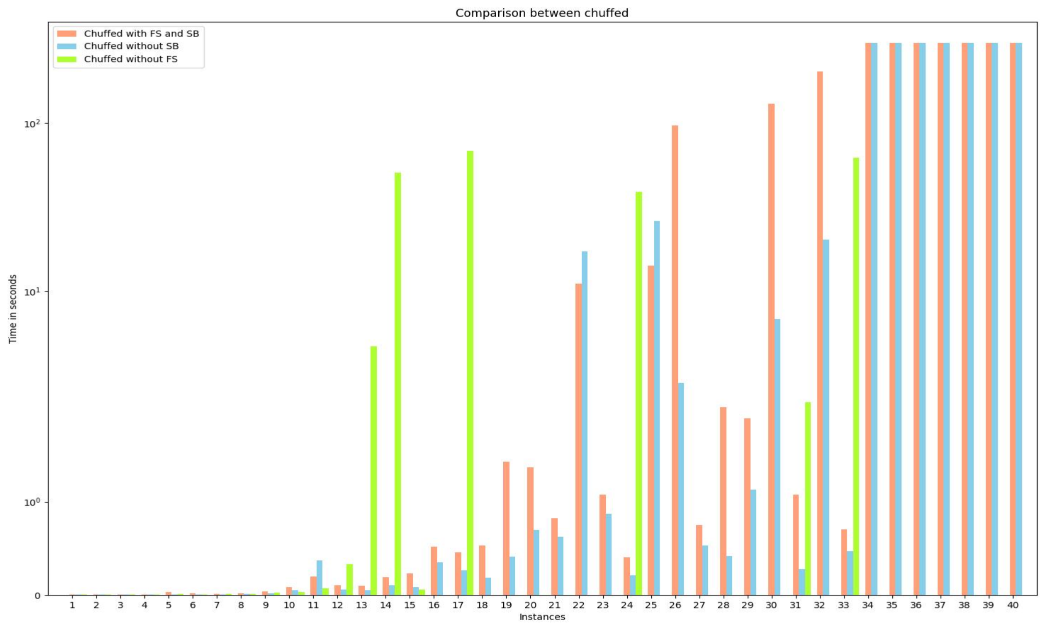


Figure 4: Comparison with chuffed

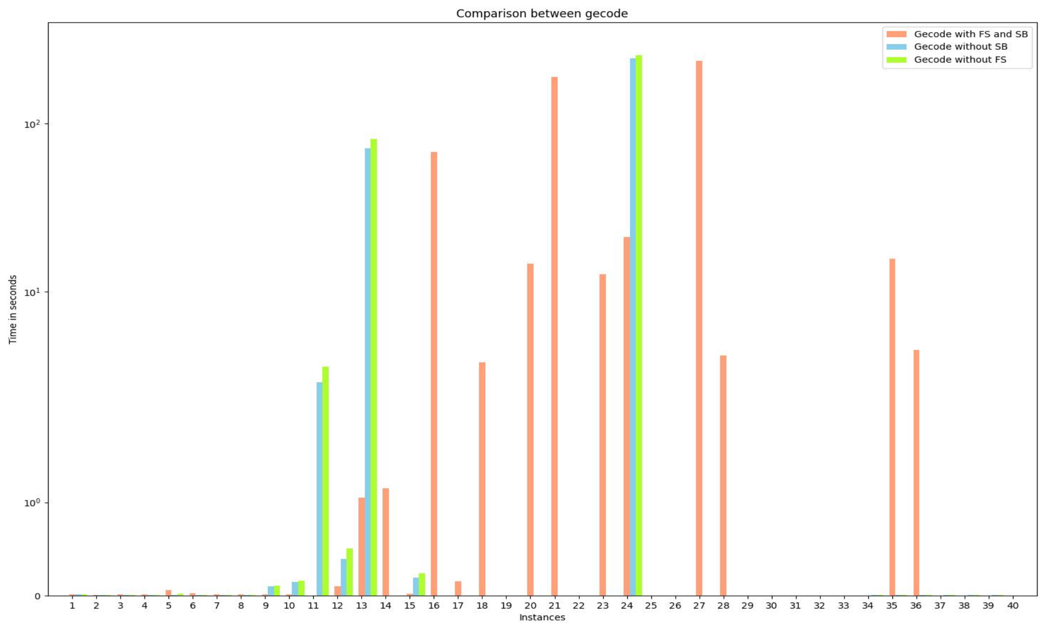


Figure 5: Comparison with gecode

Now we show the heights:

ID	Chuffed + SB + FS	Chuffed w/out SB + FS	Chuffed + SB + w/out FS	Gecode + SB + FS	Gecode w/out SB + FS	Gecode + SB w/out FS
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**	N/A	N/A
15	**22**	**22**	**22**	**22**	**22**	**22**
16	**23**	**23**	N/A	**23**	N/A	N/A
17	**24**	**24**	**24**	**24**	N/A	N/A
18	**25**	**25**	N/A	**25**	N/A	N/A
19	**26**	**26**	N/A	N/A	N/A	N/A
20	**27**	**27**	N/A	**27**	N/A	N/A
21	**28**	**28**	N/A	**28**	N/A	N/A
22	**29**	**29**	N/A	N/A	N/A	N/A
23	**30**	**30**	N/A	**30**	N/A	N/A
24	**31**	**31**	**31**	**31**	N/A	**31**
25	**32**	**32**	N/A	N/A	N/A	N/A
26	**33**	**33**	N/A	N/A	N/A	N/A
27	**34**	**34**	N/A	**34**	N/A	N/A
28	**35**	**35**	N/A	**35**	**35**	N/A
29	**36**	**36**	N/A	N/A	N/A	N/A
30	**37**	**37**	N/A	N/A	N/A	N/A
31	**38**	**38**	**38**	N/A	N/A	N/A
32	**39**	**39**	N/A	N/A	N/A	N/A
33	**40**	**40**	**40**	N/A	N/A	N/A
34	40	40	N/A	N/A	N/A	**40**
35	40	40	N/A	**40**	**40**	**40**
36	40	40	N/A	**40**	**40**	**40**
37	60	60	N/A	N/A	N/A	**60**
38	60	60	N/A	N/A	N/A	**60**
39	60	60	N/A	N/A	N/A	**60**
40	92	92	N/A	N/A	N/A	N/A

Table 1: Results using Gecode and Chuffed with and without symmetry breaking and free search **without** rotation

The following results are made by our model with rotation, with time limit of 5 minutes. Also in the rotation model, symmetry breaking and free search have a fundament role to improve the quality of the solutions.

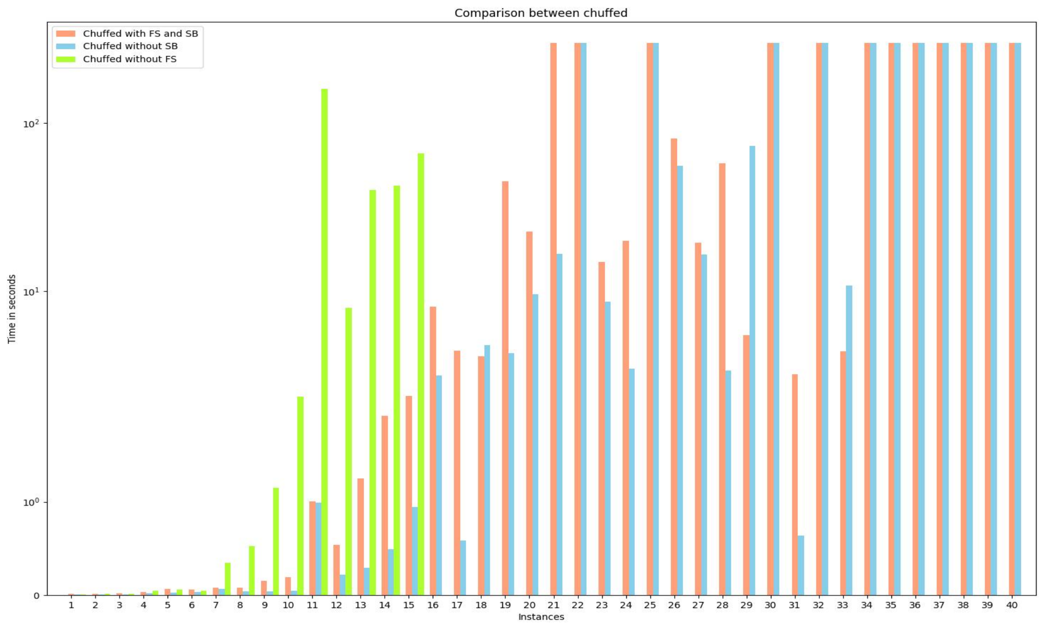


Figure 6: Comparison with chuffed

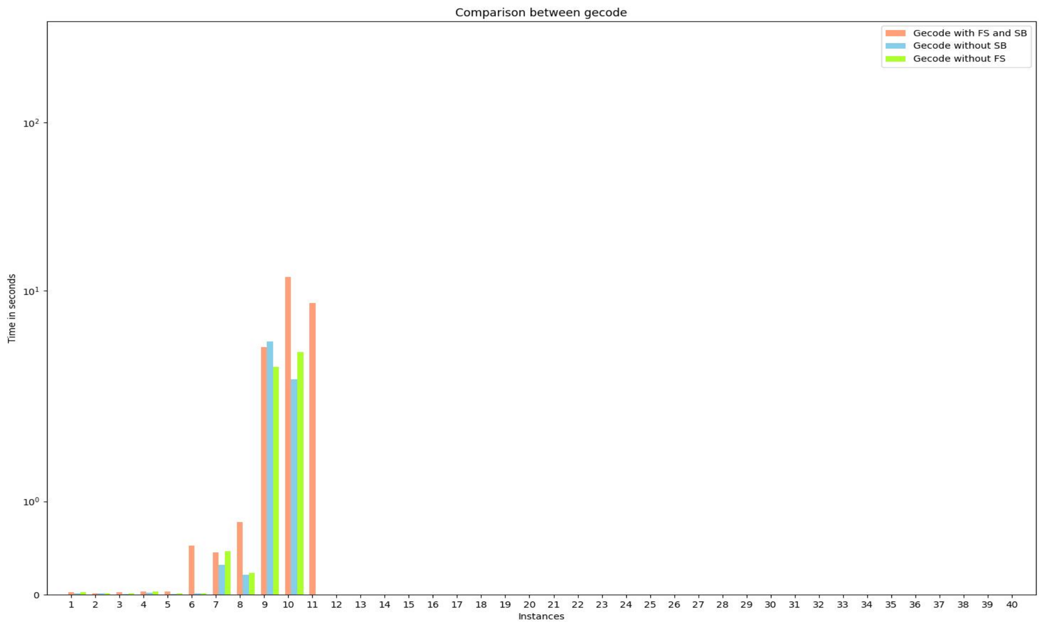


Figure 7: Comparison with gecode

ID	Chuffed + SB + FS	Chuffed w/out SB + FS	Chuffed w/out FS + SB	Gecode + SB + FS	Gecode w/out SB + FS	Gecode + SB w/out FS
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**	N/A	**18**	**18**	N/A	N/A
12	**19**	**19**	**19**	N/A	N/A	N/A
13	**20**	**20**	**20**	N/A	N/A	N/A
14	**21**	**21**	**21**	N/A	N/A	N/A
15	**22**	**22**	**22**	N/A	N/A	N/A
16	**23**	**23**	N/A	N/A	N/A	N/A
17	**24**	**24**	N/A	N/A	N/A	N/A
18	**25**	**25**	N/A	N/A	N/A	N/A
19	**26**	**26**	N/A	N/A	N/A	N/A
20	**27**	**27**	N/A	N/A	N/A	N/A
21	29	**28**	N/A	N/A	N/A	N/A
22	30	30	N/A	N/A	N/A	N/A
23	**30**	**30**	N/A	N/A	N/A	N/A
24	**31**	**31**	N/A	N/A	N/A	N/A
25	33	33	N/A	N/A	N/A	N/A
26	**33**	**33**	N/A	N/A	N/A	N/A
27	**34**	**34**	N/A	N/A	N/A	N/A
28	**35**	**35**	N/A	N/A	N/A	N/A
29	**36**	**36**	N/A	N/A	N/A	N/A
30	38	38	N/A	N/A	N/A	N/A
31	**38**	**38**	N/A	N/A	N/A	N/A
32	40	40	N/A	N/A	N/A	N/A
33	**40**	**40**	N/A	N/A	N/A	N/A
34	40	40	N/A	N/A	N/A	N/A
35	40	40	N/A	N/A	N/A	N/A
36	40	40	N/A	N/A	N/A	N/A
37	61	61	N/A	N/A	N/A	N/A
38	61	61	N/A	N/A	N/A	N/A
39	61	61	N/A	N/A	N/A	N/A
40	92	92	N/A	N/A	N/A	N/A

Table 2: Results using Gecode and Chuffed with and without symmetry breaking and free search **with** rotation

3 SMT

3.1 Decision variables

Our SMT model contains the same decisions variables already described on Section 1. The problem has width and length of the n circuits and required to find the right position in the plate to minimize the height but remaining inside the width of it. One decision variable used is the **circuit_area** that $\forall i \in \{0, n\_circuits\}$ it contains all the areas for every circuit, with the single area calculated as $x_{\text{i}} \cdot y_{\text{i}}$.

3.2 Objective function

For our SMT model, we minimize the height where $height = \max_{i \in n\_circuits} (y_{\text{i}} + coord\_y_{\text{i}})$ because our aim is to have all circuits in the smallest space. We check the maximum height given by the sum of the coordinates of the highest circuit plus his height. The bounds of the height are already described in Section 1.

3.3 Constraints

In our SMT basic model, we have imposed the following constraints:

$$
\begin{aligned}
\text{min} \quad & \textit{height} && (1)\\
\text{s.t.} \quad & \textit{height\_min} \leq \textit{height} \leq \textit{height\_max} && (2)\\
\\
& \textit{coord\_y}_{\text{i}} \geq 0 && (3)\\
& \textit{coord\_x}_{\text{i}} \geq 0 && (4)\\
& \textit{coord\_y}_{\text{i}} + y_{\text{i}} \leq \textit{height} && (5)\\
& \textit{coord\_x}_{\text{i}} + x_{\text{i}} \leq \textit{plate} && (6)\\
\\
& \textit{coord\_x}_{\text{i}} + x_{\text{i}} \leq \textit{coord\_x}_{\text{j}} \vee && (7)\\
& \textit{coord\_x}_{\text{j}} + x_{\text{j}} \leq \textit{coord\_x}_{\text{i}} \vee && (8)\\
& \textit{coord\_y}_{\text{i}} + y_{\text{i}} \leq \textit{coord\_y}_{\text{j}} \vee && (9)\\
& \textit{coord\_y}_{\text{j}} + y_{\text{j}} \leq \textit{coord\_y}_{\text{i}} && (10)\\
\\
& \sum_{i=1}^{n\_circuits} m_{\text{i}} < \textit{plate with } m_{\text{i}} = \begin{cases} x[i] & \text{if } \textit{coord\_y}[i] <= z \wedge z < \textit{coord\_y}[i] + y[i] \\ 0 & \text{otherwise} \end{cases} && (11)\\
\\
& \textit{if } x_{\text{i}} = x_{\text{j}} \wedge y_{\text{i}} = y_{\text{j}} \\
& \textit{then } \textit{coord\_x}_{\text{i}} \leq \textit{coord\_x}_{\text{j}} \wedge \textit{coord\_y}_{\text{i}} \leq \textit{coord\_y}_{\text{j}} && (12)\\
& \textit{coord\_x}_{\text{k}} = 0 \wedge \textit{coord\_y}_{\text{k}} = 0 && (13)
\end{aligned}
$$

As we said, the height must be minimized (**1**) and it's boundaries are defined in (**2**). The following constraints are defined $\forall i \in \{0, n\_circuits\}$. It's important to set the lower bound for every coord_x (**3**) and coord_y (**4**) to be greater or equal 0 and every coord_x plus his width to be inside the plate (**5**). The same goes with coord_y plus his height to be inside the height (**6**) that we are minimizing. Then, with $\forall j \in \{0, n\_circuits\} \wedge i \neq j$ the constraints (**7**), (**8**), (**9**), (**10**) are in OR condition to remove overlap, just one condition is sufficient to satisfy the constraint, since, two circuits can have the same coord_x but must be placed one before or after the other, according to y, and conversely.

The cumulative constraint (**11**) is represented by the emulation of the one used by MiniZinc. The constraint ensures that m as the sum of the widths of all circuits that overlap with a given height does not exceed the plate width at that height. The cumulative constraint is constructed by iterating over all possible heights $z \in y$ and adding a constraint for each height to ensure that the width of the circuits that overlap with that height does not exceed the plate width.

The last two constraints are useful for symmetry breaking, to not swap circuits with the same width and height (**12**) and using circuit_max_area to calculate the biggest circuit $k \in \{0, n\_circuits\}$ and place its coordinates in $(0, 0)$ to reduce the possibilities (**13**).

3.4 Rotation

In our SMT model with rotation, we have imposed the following constraints:

$$\begin{aligned}
\min \quad & height && (14)\\
\text{s.t.} \quad & height\_min \leq height \leq height\_max && (15)\\
\\
& coord\_y_{\mathrm{i}} \geq 0 && (16)\\
& coord\_x_{\mathrm{i}} \geq 0 && (17)\\
& if\ rotation_{\mathrm{i}}\ then\ rot\_y_{\mathrm{i}} = x_{\mathrm{i}} \wedge rot\_x_{\mathrm{i}} = y_{\mathrm{i}} && \\
& else\ rot\_y_{\mathrm{i}} = y_{\mathrm{i}} \wedge rot\_x_{\mathrm{i}} = x_{\mathrm{i}} && (18)\\
& coord\_x_{\mathrm{i}} + rot\_x_{\mathrm{i}} \leq plate && (19)\\
& coord\_y_{\mathrm{i}} + rot\_y_{\mathrm{i}} \leq height && (20)\\
\\
& coord\_x_{\mathrm{i}} + rot\_x_{\mathrm{i}} \leq coord\_x_{\mathrm{j}} \vee && (21)\\
& coord\_x_{\mathrm{j}} + rot\_x_{\mathrm{j}} \leq coord\_x_{\mathrm{i}} \vee && (22)\\
& coord\_y_{\mathrm{i}} + rot\_y_{\mathrm{i}} \leq coord\_y_{\mathrm{j}} \vee && (23)\\
& coord\_y_{\mathrm{j}} + rot\_y_{\mathrm{j}} \leq coord\_y_{\mathrm{i}} && (24)\\
\\
& \sum_{i=1}^{n\_circuits} m_{\mathrm{i}} < plate\ with\ m_{\mathrm{i}} \begin{cases} x[i] & \text{if } coord_y[i] \leq z \wedge z < coord_y[i] + rot_y[i] \\ 0 & \text{otherwise} \end{cases} && (25)\\
\\
& if\ x_{\mathrm{i}} = x_{\mathrm{j}} \vee y_{\mathrm{i}} = y_{\mathrm{j}} && \\
& then\ coord\_x_{\mathrm{i}} \leq coord\_x_{\mathrm{j}} \vee coord\_y_{\mathrm{i}} \leq coord\_y_{\mathrm{j}} && (26)\\
& coord\_x_{\mathrm{k}} = 0 \wedge coord\_y_{\mathrm{k}} = 0 && (27)\\
\\
& if\ x_{\mathrm{i}} = y_{\mathrm{i}}\ then\ rotation_{\mathrm{i}} = False && (28)\\
& if\ y_{\mathrm{i}} > plate\ then\ rotation_{\mathrm{i}} = False && (29)
\end{aligned}$$

For the rotation model, we add three variables ***rotation***, ***rot_x*** and ***rot_y*** already described in Section 1. Like the no rotation model, we must minimize the height (**14**) and the constraint (**15**), (**16**) and (**17**) are the same. As the no rotation model, the following constraints are defined $\forall i \in \{0, n\_circuits\}$. The constraint (**18**) let us rotate the circuit assigning $\mathrm{x_i}$ to $\mathrm{rot\_y_i}$, and conversely, if the i^{th} circuit is rotated. If the circuit is not rotated, then rot_x and rot_y will be assigned to the same x and y, respectively.

So the constraint from (**19**) to (**27**) are the same but they will use rot_x and rot_y for the respective width and height. In case of the (**26**) AND are substituted with OR to relax the model. The last two are exclusive to the rotation model, the first (**28**) does not rotate a circuit if it's height and width are the same because it doesn't make sense to rotate a square and the second (**29**) does not rotate a circuit if it's original height is longer than the plate, because it will not fit horizontally.

3.5 Validation

Experimental design To do this project, we decided to utilize Pycharm. First, it need the z3 library so it is necessary to run the command ***!pip install z3-solver***. To run it, firstly you need to change the path of the project. On the terminal we run the relative ***compiler_*.py***. We've used the solver **z3** with the z3 library for python. Running the compiler, if it's the one without rotation it will produce the results inside solution/model, while if it's the one with rotation, the folder will be solution/rotation. We kept the folders with no symmetry to compare but running the compiler will overwrite the content inside model (or rotation).

Experimental results The following results are made by our model with and without rotation, with and without symmetry breaking contraint, with time limit of 300 seconds.

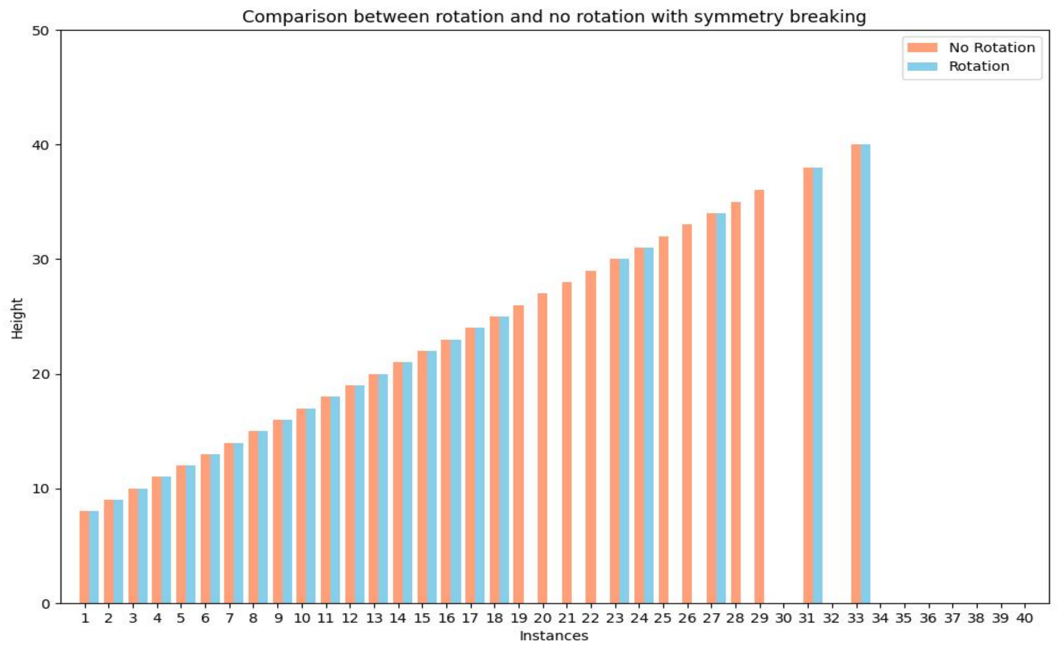


Figure 8: Comparison between rotation and no rotation with Symmetry

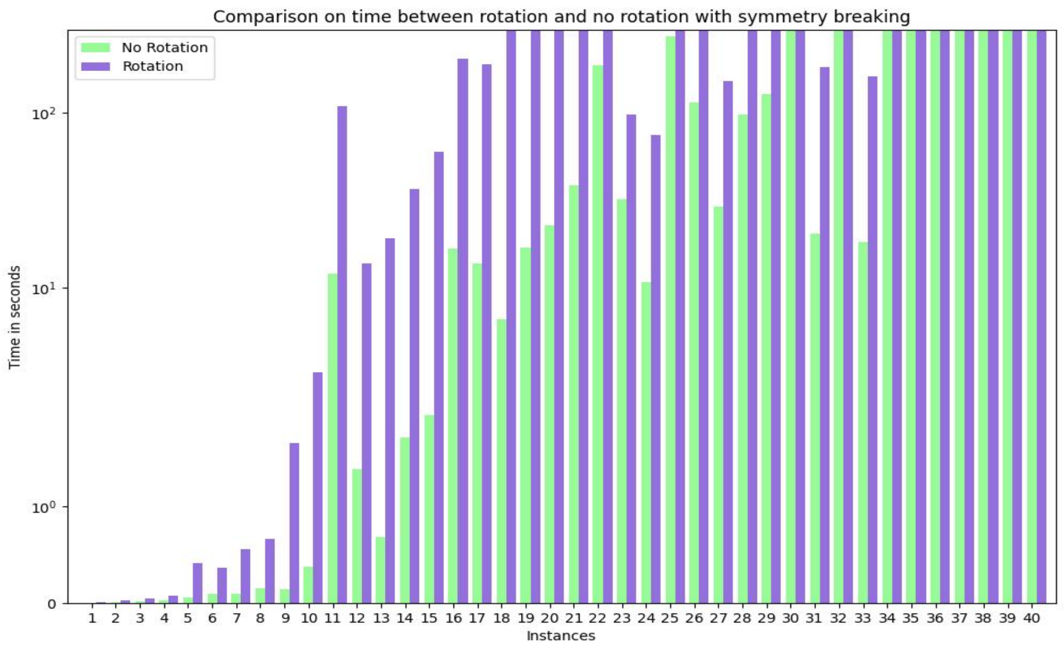


Figure 9: Comparison on time between rotation and no rotation with Symmetry

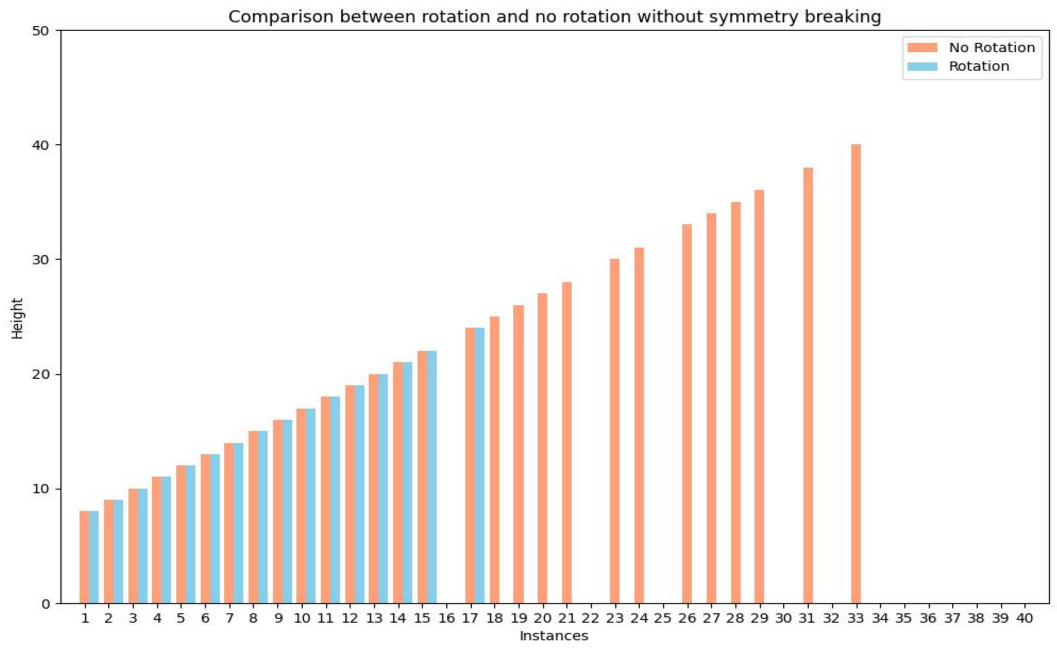


Figure 10: Comparison between rotation and no rotation without Symmetry

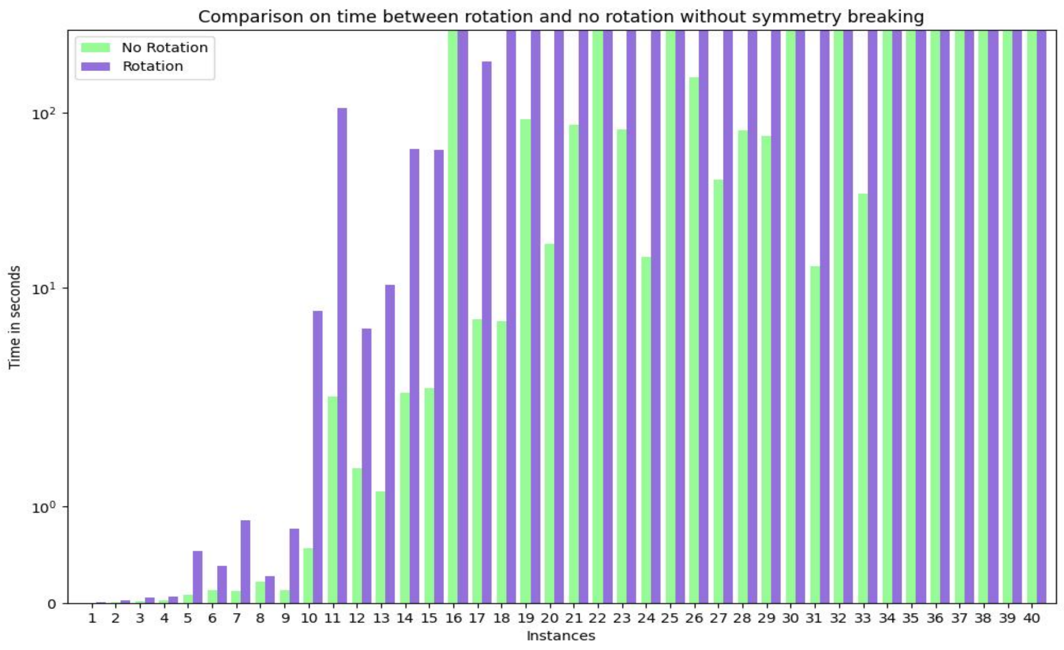


Figure 11: Comparison on time between rotation and no rotation without Symmetry

ID	No Rotation w/ symmetry	No Rotation	Rotation w/ symmetry	Rotation
1	**8**	**8**	**8**	**8**
2	**9**	**9**	**9**	**9**
3	**10**	**10**	**10**	**10**
4	**11**	**11**	**11**	**11**
5	**12**	**12**	**12**	**12**
6	**13**	**13**	**13**	**13**
7	**14**	**14**	**14**	**14**
8	**15**	**15**	**15**	**15**
9	**16**	**16**	**16**	**16**
10	**17**	**17**	**17**	**17**
11	**18**	**18**	**18**	**18**
12	**19**	**19**	**19**	**19**
13	**20**	**20**	**20**	**20**
14	**21**	**21**	**21**	**21**
15	**22**	**22**	**22**	**22**
16	**23**	N/A	**23**	N/A
17	**24**	**24**	**24**	**24**
18	**25**	**25**	**25**	N/A
19	**26**	**26**	N/A	N/A
20	**27**	**27**	N/A	N/A
21	**28**	**28**	N/A	N/A
22	**29**	N/A	N/A	N/A
23	**30**	**30**	**30**	N/A
24	**31**	**31**	**31**	N/A
25	**32**	N/A	N/A	N/A
26	**33**	**33**	N/A	N/A
27	**34**	**34**	**34**	N/A
28	**35**	**35**	N/A	N/A
29	**36**	**36**	N/A	N/A
30	N/A	N/A	N/A	N/A
31	**38**	**38**	**38**	N/A
32	N/A	N/A	N/A	N/A
33	**40**	**40**	**40**	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: The best result obtained for each instance with and without rotation, each with and without symmetry breaking.

The hardware used is the same in section 2.5.

4 MIP

4.1 Decision variables

Our MIP model contains the same decisions variables already described on Section 1. In addiction, we had to add 2 more variables:

- **area**. It is an array of variables. The bounds are [**area_min**, **area_max**] where **area_min** is the product between the actual height the model is considering and the plate and **area_max** is the product between the height_max and the plate. This variable was very useful, due that, without this, all the solutions contained circuits with space between them. This reduces the problem.

- **big_m**. It is an array of variables of boolean. It was necessary to build the constraint of no overlap of circuits.

4.2 Objective function

For our MIP model, we minimize the height because our aim is to have all circuits in the smallest space. The bounds of the height are already described on Section 1.

4.3 Constraints

In our basic model, we have imposed the following constraints:

$$\min \quad \mathit{height} \tag{30}$$

$$\text{s.t.} \quad \mathit{coord\_x}_{\mathrm{i}} + x_{\mathrm{i}} \leq \mathit{plate} \tag{31}$$

$$\mathit{coord\_y}_{\mathrm{i}} + y_{\mathrm{i}} \leq \mathit{height} \tag{32}$$

$$\mathit{coord\_x}_{\mathrm{i}} + x_{\mathrm{i}} \leq \mathit{coord\_x}_{\mathrm{j}} + (\mathit{big\_m}_{\mathrm{i,j,0}}) \cdot \mathit{plate} \tag{33}$$

$$\mathit{coord\_y}_{\mathrm{i}} + y_{\mathrm{i}} \leq \mathit{coord\_y}_{\mathrm{j}} + (\mathit{big\_m}_{\mathrm{i,j,1}}) \cdot \mathit{height\_max} \tag{34}$$

$$\mathit{coord\_x}_{\mathrm{j}} + x_{\mathrm{j}} \leq \mathit{coord\_x}_{\mathrm{i}} + (\mathit{big\_m}_{\mathrm{i,j,2}}) \cdot \mathit{plate} \tag{35}$$

$$\mathit{coord\_y}_{\mathrm{j}} + y_{\mathrm{j}} \leq \mathit{coord\_y}_{\mathrm{i}} + (\mathit{big\_m}_{\mathrm{i,j,3}}) \cdot \mathit{height\_max} \tag{36}$$

$$\sum_{k=0}^{3} \mathit{big\_m}_{\mathrm{i,j,k}} \leq 3 \tag{37}$$

$$\mathit{area} = \mathit{height} \cdot \mathit{plate} \tag{38}$$

$$\mathit{area} \geq \mathit{area\_min} \tag{39}$$

$$\mathit{area} \leq \mathit{area\_max} \tag{40}$$

The first problem that came us in mind is that we need that the circuits have not to be over the plate or the height. For that, we imposed the constraints (**31**) and (**32**).

Then another problem showed up: the overlapping. To delete this issue, we defined the constraints from (**33**) to (**36**) by using BIG_M technique. So, the i^{th} and the j^{th} circuits don't overlap if least one of the constraints are satisfied. The first and the third constraint represent the case when the i^{th} and j^{th} circuit are side by side. The second and the fourth represent when one of them is below the other. The constraint (**37**) is applied because if any constraint is satisfied, then the circuits overlap.

Another problem continued to persist on our solution: the space between the circuits. To reduce it, we added three constraints from (**38**) to (**40**), that include a new variable that is the area. This, allows us, to have a better solution.

4.4 Rotation

$$
\begin{aligned}
\min \quad & \mathit{height} && (41)\\
\text{s.t.} \quad & \mathit{coord\_x}_{\mathrm{i}} + x_{\mathrm{i}} \le \mathit{plate} && (42)\\
& \mathit{coord\_y}_{\mathrm{i}} + y_{\mathrm{i}} \le \mathit{height} && (43)\\
\\
& \mathit{coord\_x}_{\mathrm{i}} + x_{\mathrm{i}} \cdot (1 - \mathit{rotation}_{\mathrm{i}}) + y_{\mathrm{i}} \cdot \mathit{rotation}_{\mathrm{i}} \le \mathit{coord\_x}_{\mathrm{j}} + (\mathit{big\_m}_{\mathrm{i,j,0}}) \cdot \mathit{plate} && (44)\\
& \mathit{coord\_y}_{\mathrm{i}} + y_{\mathrm{i}} \cdot (1 - \mathit{rotation}_{\mathrm{i}}) + x_{\mathrm{i}} \cdot \mathit{rotation}_{\mathrm{i}} \le \mathit{coord\_y}_{\mathrm{j}} + (\mathit{big\_m}_{\mathrm{i,j,1}}) \cdot \mathit{height\_max} && (45)\\
& \mathit{coord\_x}_{\mathrm{j}} + x_{\mathrm{j}} \cdot (1 - \mathit{rotation}_{\mathrm{j}}) + y_{\mathrm{j}} \cdot \mathit{rotation}_{\mathrm{j}} \le \mathit{coord\_x}_{\mathrm{i}} + (\mathit{big\_m}_{\mathrm{i,j,2}}) \cdot \mathit{plate} && (46)\\
& \mathit{coord\_y}_{\mathrm{j}} + y_{\mathrm{j}} \cdot (1 - \mathit{rotation}_{\mathrm{j}}) + x_{\mathrm{j}} \cdot \mathit{rotation}_{\mathrm{j}} \le \mathit{coord\_y}_{\mathrm{i}} + (\mathit{big\_m}_{\mathrm{i,j,3}}) \cdot \mathit{height\_max} && (47)\\
& \sum_{k=0}^{3} \mathit{big\_m}_{\mathrm{i,j,k}} \le 3 && (48)\\
\\
& \mathit{area} = \mathit{height} \cdot \mathit{plate} && (49)\\
& \mathit{area} \ge \mathit{area\_min} && (50)\\
& \mathit{area} \le \mathit{area\_max} && (51)\\
\\
& x_{\mathrm{i}} = y_{\mathrm{i}} \rightarrow \mathit{rotation}_{\mathrm{i}} = 0 && (52)\\
& y_{\mathrm{i}} > \mathit{plate} \rightarrow \mathit{rotation}_{\mathrm{i}} = 0 && (53)
\end{aligned}
$$

For the rotation model, all the constraints are the same of the other model but the overlap one. In fact, we had to consider the new dimensions of the circuit. In the overlap constraint from (**44**) to (**47**) it is added the rotation array, so if the circuit is rotated, it will use its width as its height and conversely.

Then we added 2 constraints that certainly helps the search of the solution. We force to not rotate the circuits that have the same height and width (**52**), because it is useless to rotate a square, and if the height of the circuit is greater than the plate (**53**), because it will not respect the constraint (**42**).

4.5 Validation

Experimental design To do this project, we decided to utilize Pycharm. On the terminal we run ***compiler.py***. It will automatically run the no rotation model and the one with rotation. To run it, firstly you need to change the path of the project. On src/MIP/py directory, you can find compiler.py. In the compiler method there's a string called PATH which you can modify. Below that you can modify also the number of instance (from_instance and to_instance). To change the solver you must change the name of the solver in the prob.solver() that is in both methods model_mip and model_mip_rotation.

The hardware used is the same in section 2.5.

Experimental results For MIP model, we decided to test three solvers:

- **CPLEX**. We chose this solver because we have seen would have give promising results.
- **PULP_CBC_CMD**. We chose this solver because to model this problem we used pulp library.
- **GUROBI**. We chose this solver due to its capabilities

For all the solver we plotted the time and the height comparing them. We show only the once with time (for the heights we designed the tables).

As you can see in the figure 13, GUROBI is able to solve the task in much less time and the solutions found are better than CPLEX and PULP, on the other hand it is not able to solve 10 instances. CPLEX, instead, is the middle ground in terms of time and solutions found. PULP is the solver that takes more time and the solutions are not even near to the optimal ones.

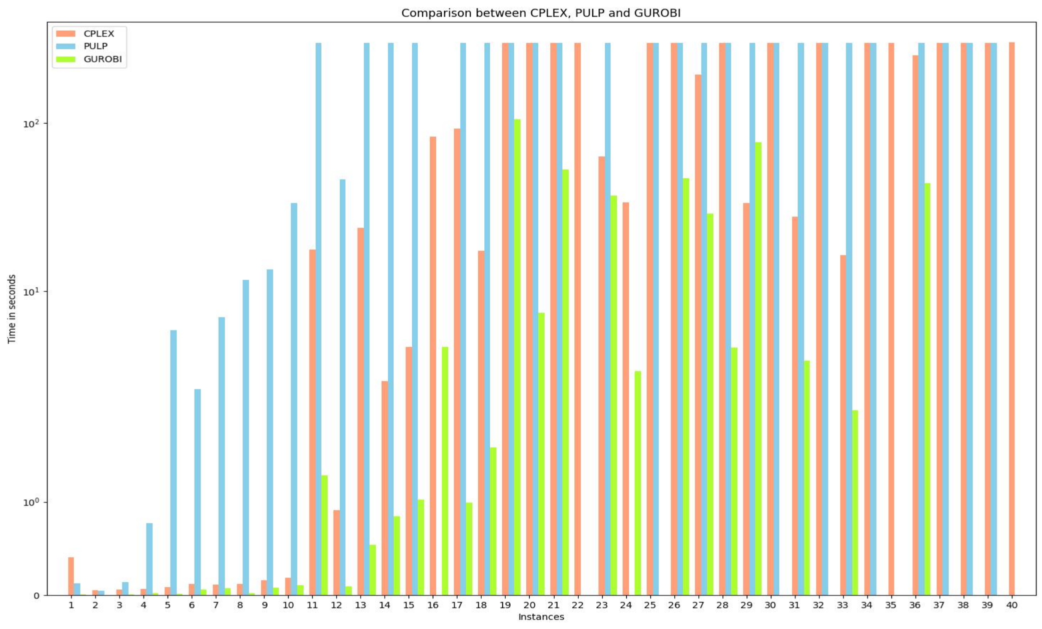


Figure 12: Comparison between CPLEX, PULP and GUROBI on time

Now we show the heights:

ID	CPLEX	PULP_CBC_CMD	GUROBI
1	**8**	**8**	**8**
2	**9**	**9**	**9**
3	**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**
11	**18**	**35**	**18**
12	**19**	**19**	**19**
13	**20**	**26**	**20**
14	**21**	**36**	**21**
15	**22**	**37**	**22**
16	**23**	N/A	**23**
17	**24**	**47**	**24**
18	**25**	**50**	**25**
19	**27**	**42**	**26**
20	**28**	**44**	**27**
21	**29**	**74**	**28**
22	**30**	N/A	N/A
23	**30**	**48**	**30**
24	**31**	N/A	**31**
25	**33**	**36**	N/A
26	**34**	**41**	**33**
27	**34**	**37**	**34**
28	**36**	**48**	**35**
29	**36**	**39**	**36**
30	**38**	**42**	N/A
31	**38**	**49**	**38**
32	**42**	**73**	N/A
33	**40**	**55**	**40**
34	**42**	**51**	N/A
35	**41**	N/A	N/A
36	**40**	**86**	**40**
37	**62**	**80**	N/A
38	**62**	**77**	N/A
39	**61**	**72**	N/A
40	**123**	N/A	N/A

Table 4: Result obtained for each instance without rotation with CPLEX and PULP

In the table 4, you can see that the best performance was made by GUROBI in the instance 19-20-21-28 in respect to CPLEX.

In the end, we did the same for the rotation model, where all the solvers have the same behavior respect the other model.

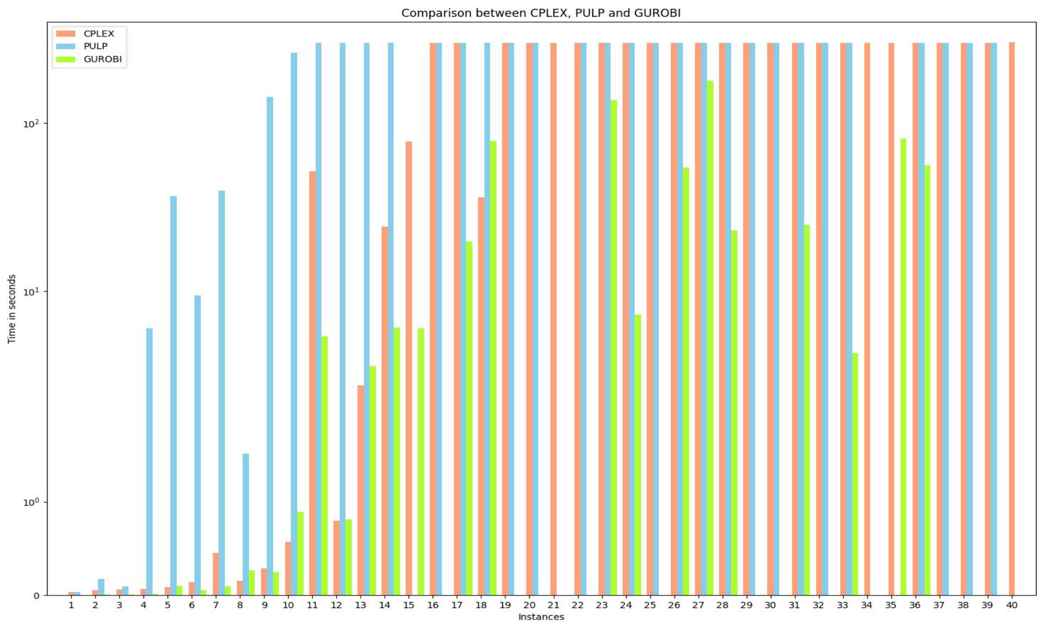


Figure 13: Comparison between CPLEX, PULP and GUROBI on time

Now we show the heights:

ID	CPLEX	PULP_CBC_CMD	GUROBI
1	**8**	**8**	**8**
2	**9**	**9**	**9**
3	**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**
11	**18**	**45**	**18**
12	**19**	**30**	**19**
13	**20**	**25**	**20**
14	**21**	**34**	**21**
15	**22**	N/A	**22**
16	**24**	**35**	N/A
17	**25**	**75**	**24**
18	**25**	**42**	**25**
19	**27**	**104**	N/A
20	**28**	**61**	N/A
21	**29**	N/A	N/A
22	**31**	**116**	N/A
23	**31**	**47**	**30**
24	**32**	**108**	**31**
25	**33**	**47**	N/A
26	**34**	**35**	**33**
27	**35**	**36**	**34**
28	**36**	**43**	**35**
29	**37**	**39**	N/A
30	**39**	**44**	N/A
31	**39**	**46**	**38**
32	**41**	**106**	N/A
33	**41**	**74**	**40**
34	**41**	N/A	N/A
35	**41**	N/A	**40**
36	**41**	**50**	**40**
37	**62**	**72**	N/A
38	**62**	**76**	N/A
39	**61**	**70**	N/A
40	**103**	N/A	N/A

Table 5: Result obtained for each instance with rotation with CPLEX and PULP

5 Conclusions

To conclude, we have chosen to show all the results with the best solvers for each model. For CP we chose chuffed and for MIP CPLEX. We are satisfied from our models' results. The one the has the higher improve is certainly CP, that not only solves all the instances but is able to find better solutions than SMT and MIP. For SMT it's clear that the task take longer and it could probably satisfy all the instances but in a larger period of time.

ID	CP	SMT	MIP
1	**8**	**8**	**8**
2	**9**	**9**	**9**
3	**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**
11	**18**	**18**	**18**
12	**19**	**19**	**19**
13	**20**	**20**	**20**
14	**21**	**21**	**21**
15	**22**	**22**	**22**
16	**23**	**23**	**23**
17	**24**	**24**	**24**
18	**25**	**25**	**25**
19	**26**	**26**	**27**
20	**27**	**27**	**28**
21	**29**	**28**	**29**
22	**30**	**29**	**30**
23	**30**	**30**	**30**
24	**31**	**31**	**31**
25	**32**	**32**	**33**
26	**33**	**33**	**34**
27	**34**	**34**	**34**
28	**35**	**35**	**36**
29	**36**	**36**	**36**
30	**37**	N/A	**38**
31	**38**	**38**	**38**
32	**39**	N/A	**42**
33	**40**	**40**	**40**
34	40	N/A	**42**
35	40	N/A	**41**
36	40	N/A	**40**
37	60	N/A	**62**
38	60	N/A	**62**
39	60	N/A	**61**
40	92	N/A	**123**

Table 6: The best result obtained for each instance without rotation.

For the rotation model, we had not results as great as the model without rotation. Anyway, CP still has the best results.

ID	CP	SMT	MIP
1	**8**	**8**	**8**
2	**9**	**9**	**9**
3	**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**
11	**18**	**18**	**18**
12	**19**	**19**	**19**
13	**20**	**20**	**20**
14	**21**	**21**	**21**
15	**22**	**22**	**22**
16	**23**	**23**	**24**
17	**24**	**24**	**25**
18	**25**	**25**	**25**
19	**26**	N/A	**27**
20	**27**	N/A	**28**
21	29	N/A	**29**
22	30	N/A	**31**
23	**30**	**30**	**31**
24	**31**	**31**	**32**
25	33	N/A	**33**
26	**33**	N/A	**34**
27	**34**	**34**	**35**
28	**35**	N/A	**36**
29	**36**	N/A	**37**
30	38	N/A	**39**
31	**38**	**38**	**39**
32	40	N/A	**41**
33	**40**	**40**	**41**
34	40	N/A	**41**
35	40	N/A	**41**
36	40	N/A	**41**
37	61	N/A	**62**
38	61	N/A	**62**
39	61	N/A	**61**
40	92	N/A	**103**

Table 7: The best result obtained for each instance with rotation.