Alma Mater Studiorum - Artificial Intelligence

Combinatorial Decision Making and Optimization

CP, SMT and MIP approaches to the VLSI problem

Francesca Boccardi - francesca.boccardi@studio.unibo.it

Luigi Podda - luigi.podda@studio.unibo.it

Contents

Introduction

In recent years, Very Large-Scale Integration (VLSI) has been a well known problem: it is the process of organizing millions or even billions of electronic components in order to fit more of them in a given area.
Considering the increasing number of components that are involved, it is necessary to address these problems with smart and efficient techniques.

This report describes three different Combinatorial Optimization approaches to the VLSI problem:

- Constraint Programming (CP)
- Satisfiability Modulo Theories (SMT)
- Mixed Integer-Linear Programming (MIP)

Problem formulation

In this particular case, the aim of the problem is to fit a set of n circuits, having width and height in r and d arrays respectively, into a plate with a fixed width w. Beyond the base version of the problem, where all circuits have fixed height and width, an other variety of it has been considered, allowing to obtain a solution with rotated circuits, namely with swapped original dimensions. The objective is in any case to find each circuit position on the board without exceeding the given width and minimizing its height.

Input

As input there are different integers values. Below there is an example of the input format as it is in the input files.

$w = 8$;
$n = 4$;
$d = [3, 5, 3, 5]$;
$r = [3, 3, 5, 5]$;

The explanation of each parameter can be found in the "Parameters" section for each of the three problem approaches.

Output

For all the instances the solver managed to optimally solve within the time limit, namely 300s, the output is formatted in the following way:

8 8
4
3 3 0 5
3 5 5 0
5 3 3 5
5 5 0 0
———-
==========
0.32069949992001057

In particular, in the first row there are two integers, respectively the width and the height of the plate.
In the second row there is the number n of circuit in the instance.

In the following n rows there are the width, the height, the horizontal coordinate and the vertical coordinate of each circuit.
The last row shows the total amount of time employed by the solver to reach the optimal solution.

Common preprocessing

For efficiency reasons, all three employed techniques apply a domain reduction on *height* values, by defining a proper upper bound. In particular, the maximum plate height is expressed as h_{max} and it is computed as

$$h_{max} = \sum_{i=1}^{L} d\_sort_i$$

where d_sort contains all d elements in descending order and $L = min(n, levels)$, having

$$levels = \left(\left\lfloor \frac{\sum_{i=1}^{n} r_i}{w} \right\rfloor + 1 \right) \cdot 2$$

Going into details, $levels$ is determined approximately calculating how many "rows" of length w need to be vertically displaced such that all circuits can be distributed horizontally along these rows. Then, h_{max} is found summing the heights of the L highest circuits, taking L as the minimum between the number of circuits n and $levels$.
Actually, h_{max} is computed taking into account the worst case scenario into which the L highest circuits are all located along different rows. Must be noticed that, if for approximation reasons $n < levels$, h_{max} is nothing but the sum of all circuits heights.

CP model

The first technique used for the Combinatorial Optimization approach to the VLSI problem is Constraint Programming. CP is a declarative programming paradigm where a solver finds a solution, if it exists, by assigning to each variable a value, satisfying all the constraints previously defined.

Parameters

Concerning the constant quantities defined in the model, the notation used is as follows:

- $w \rightarrow$ width of the plate
- $r_i \rightarrow$ original width of the i_{th} circuit
- $d_i \rightarrow$ original height of the i_{th} circuit
- $n \rightarrow$ total number of circuits
- $h_{max} \rightarrow$ maximum plate height, computed as explained in Common preprocessing section.
- $h_{min} \rightarrow$ minimum plate height, computed as

  $$h_{min} = \frac{\sum_{i=1}^{n} r_i \cdot d_i}{w}$$

  Indeed, the denominator is the minimum total area, namely the sum of all circuits areas. Supposing to be able to place all circuits such that the actual total area is equal to the minimum one, the height of the plate would simply be the area's value over the width of the plate, which is exactly h_{min}.
- $area_{max} \rightarrow$ maximum area of the whole plate, computed as

  $$area_{max} = h_{max} \cdot w$$
- $area_{min} \rightarrow$ minimum area of the plate, computed as

  $$area_{min} = \sum_{i=1}^{n} r_i \cdot d_i$$

Variables

No rotation model

The CP formalization of the no-rotation model is based on the definition of two arrays of variables of size n:

- $x \rightarrow$ bottom left x coordinates of circuits, bounded between 0 and x_{max}, having

  $$x_{max} = w - \min_{i \in \{1..n\}} (r_i)$$
- $y \rightarrow$ bottom left y coordinates of circuits, bounded between 0 and y_{max}, having

  $$y_{max} = h_{max} - \min_{i \in \{1..n\}} (d_i)$$

and the single variables

- $area \rightarrow$ plate's area, bounded between $area_{min}$ and $area_{max}$ and defined as

  $$height \cdot w$$
- $height \rightarrow$ height of the plate, bounded between h_{min} and h_{max}, to be minimized

Rotation model

Regarding the rotation model, the variables that have been used are exactly the same of the model without rotations. The only difference is that it is necessary to add two sets of n variables, dx and dy, which model respectively width and height of single circuits:

- $dx \rightarrow$ actual widths of circuits, bounded between 1 and dx_{max}, having

$$dx_{max} = \max\{\max_{i \in \{1..n\}}(d_i), \max_{i \in \{1..n\}}(r_i)\}$$

- $dy \rightarrow$ actual height of circuits, bounded between 1 and dy_{max}, having

$$dy_{max} = \max\{\max_{i \in \{1..n\}}(d_i), \max_{i \in \{1..n\}}(r_i)\}$$

Indeed, the rotation of a circuit is handled as a swap between its width and height, making necessary the modelling of circuits dimensions as decision variables. As a consequence, upper bounds of x and y variables must be redefined, namely:

$$x_{max} = w - \min\{\min_{i \in \{1..n\}}(d_i), \min_{i \in \{1..n\}}(r_i)\}$$

$$y_{max} = h_{max} - \min\{\min_{i \in \{1..n\}}(d_i), \min_{i \in \{1..n\}}(r_i)\}$$

Also, an additional array of n boolean variables must be defined, namely *rotation*. These variables are used because it is unknown if in the solution a circuit will be rotated or not. Thus, it is necessary to have some variables that express the rotation of circuits.

- *rotation* $\rightarrow$ array of boolean variables, where each of them is `true` if in the solution the correspondent circuit is rotated with respect its original dimensions, `false` otherwise

Constraints

No rotation model

The first two constraints are defined in order to prevent the placement of circuits from exceeding the *height* and the *width* of the plate.

$$\forall i \in \{1..n\} \quad y_i + d_i \leq height \tag{1}$$

$$\forall i \in \{1..n\} \quad x_i + r_i \leq w \tag{2}$$

The `cumulative(s, d, r, b)` global constraint has been used both horizontally and vertically. This constraint 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. Following this parallelism with a scheduling global constraint, it has been possible to find the circuits' coordinates in order to not exceed the width horizontally and the height vertically.

$$cumulative(y, d, r, w) \tag{3}$$

$$cumulative(x, r, d, height) \tag{4}$$

The `diffn` global constraint, given circuits coordinates and dimensions, has been used in order to prevent them from overlapping.

$$diffn(x, y, r, d) \tag{5}$$

With the `lex_lesseq` global constraint it has been possible to break horizontal symmetries. In particular, this symmetry breaking constraint allows to exclude all those solutions where x and r take values that make $w - x - r$ elements lexicographically smaller then x.

$$\forall i \in \{1..n\} \quad lex\_lesseq(x_i, w - x_i - r_i) \tag{6}$$

Rotation model

As previously described, this model recovers all the constraints from the no-rotation one, but modelling width and height of each circuit with dx and dy variables respectively, in order to enable rotations.
The defined constraints for this model are:

$$\forall i \in \{1..n\} \quad y_i + dy_i \leq height \tag{7}$$
$$\forall i \in \{1..n\} \quad x_i + dx_i \leq w \tag{8}$$
$$cumulative(y_{i..n}, dy_{i..n}, dx_{i..n}, w) \tag{9}$$
$$cumulative(x_{i..n}, dx_{i..n}, dy_{i..n}, height) \tag{10}$$
$$diffn(x_{i..n}, y_{i..n}, dx_{i..n}, dy_{i..n}) \tag{11}$$
$$\forall_{i \in \{1..n\}} \quad lex(x, w - x_i - dx_i) \tag{12}$$

In addition, it is necessary to introduce a new constraint in such a way that if the i_{th} *rotation* variable takes value `true`, in the solution the correspondent circuit is rotated with respect its original dimensions. More precisely, if the correspondent circuit has to be rotated, its dimensions will be swapped, having the width as the height and the height as the width. Formally:

$$\begin{aligned} \forall i \in \{1..n\} \quad & if \quad rotation_i \\ & then \quad dy_i = r_i \wedge dx_i = d_i \\ & else \quad dy_i = d_i \wedge dx_i = r_i \end{aligned} \tag{13}$$

Objective function

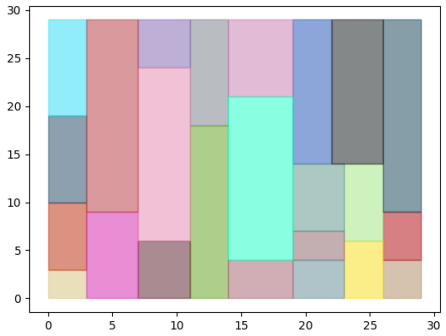
According to the VLSI problem, the quantity needed to be optimize is the *height* of the plate. Thus, the plate *height* is modelled as a decision variable and it's linked by definition to the y coordinates of circuits, which are also indirectly influenced by the minimization. However its domain has been restricted by an upper as well as a lower bound, in order to make the search more efficient.

Solver and search strategy

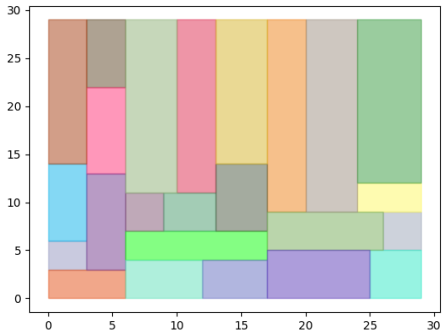
A critical part in the design choices of the models was the search strategy. Trying different combinations of search annotation variables, it has been found that the best approach was searching firstly on x and y and finally on *height*.
Moreover, the best variable choice annotation turned out to be `first_fail` for both x and y, leading the solver to choose the variable with the smallest current domain size. For *height* was enough to set the `input_order` option, as it is a single variable and there are no points choice.
Finally, `indomain_min` was set as the way to constraint all 3 set of variables, which allows to assign them the smallest domain value.
Trying different solvers and search heuristics it has been realized that the combination between the Chuffed solver, allowing the free search option, and the previously described design choices show the best results for both the models.

Results and performances

All the 40 instances have been tested applying both models. As expected, the rotation model turned out to be harder to solve with respect to the no-rotation one, as it is more constrained and it requires the definition of additional variables, and the performances were different between the two.
In particular, with a timeout of 300s, the no-rotation model solved a lager number of instances employing a shorter amount of time for each of them.
As an example, it is interesting to visualize the two different solutions found by the solver to the 22^{th} instance when applying the two models. Both of them lead to an optimal solution, which is however different between the two models, as it can be observed from the plots below:



(a) No-rotation model optimal solution



(b) Rotation model optimal solution

Figure 1: Comparison between different models solutions to instance 22

The barplot below shows all solving times, employing marked colors for those instances whose optimal solution was found within 300s, while shading tonality for those for which the solver ran out of time.

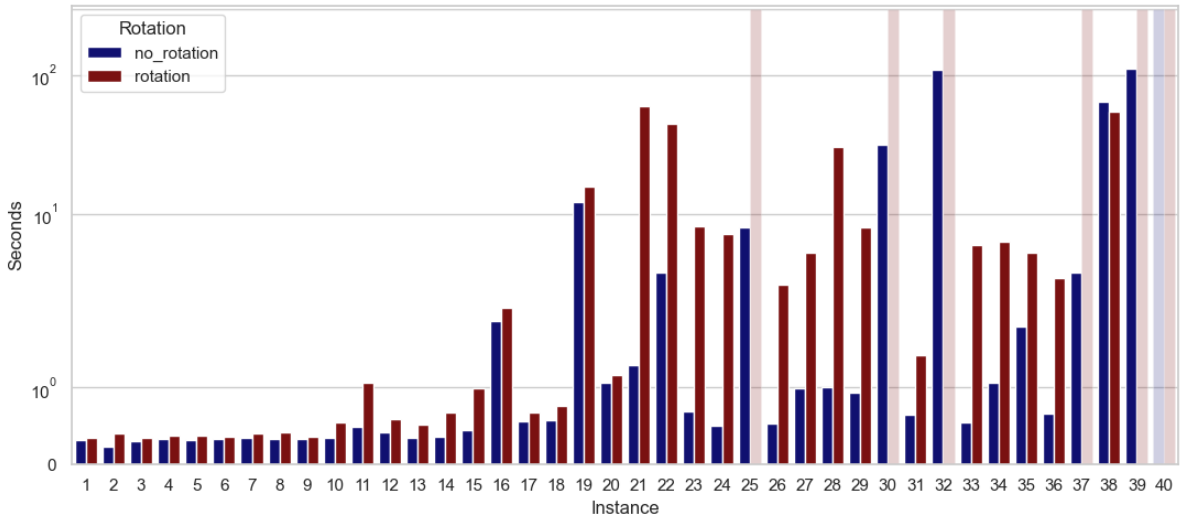


Figure 2: Barplot of CP solving times

To better analyse solving times, all of them have been collected in the following table:

	No rotation	Rotation
Instance	Solving time(s)	Solving time (s)
1	0.321	0.345
2	0.238	0.407
3	0.31	0.345
4	0.332	0.373
5	0.322	0.373
6	0.322	0.362
7	0.346	0.407
8	0.328	0.415
9	0.333	0.366
10	0.34	0.549
11	0.489	1.062
12	0.421	0.588
13	0.352	0.52
14	0.364	0.68
15	0.447	0.994
16	1.868	2.085
17	0.56	0.674
18	0.582	0.764
19	12.154	15.689
20	1.065	1.166
21	1.294	59.318
22	3.772	44.769
23	0.691	8.157
24	0.504	7.092
25	7.946	-
26	0.538	3.101
27	0.995	5.242
28	1.002	30.295
29	0.937	7.95
30	31.328	-
31	0.651	1.414
32	108.528	-
33	0.547	5.983
34	1.063	6.309
35	1.792	5.223
36	0.662	3.408
37	3.756	-
38	64.374	54.809
39	112.025	-
40	-	-

Figure 3: Table with CP solving times details

Finally, the no-rotation model clearly showed better performances, both in terms of solving times and number of instances solved.

SMT model

Another optimization technique used to solve the VLSI problem is the Satisfiability Modulo Theory, which concerns the satisfiability of logical formulas with respect to some background theories. In particular, SMT deals with first-order logic expressions and with symbols whose interpretation is constrained by a specific background theory.

Preprocessing

Before applying the model, it turned out to be very efficient to add some manipulations on the available instances data.
In particular, sorting the set of circuits according to a descending order of their area values, the SMT solver leads to better results, solving a larger number of instances in a smaller amount of time.
This kind of behaviour can be possibly be explained by the fact that it is harder to manage bigger circuits with respect to smaller ones, as moving large blocks means to deal with changes in a larger space of the plate. Thus, harder circuits should be placed first, then properly allocating the others.

Parameters

Before feeding them into the model, instances data are manipulated as explained above, obtaining:

- $w \rightarrow$ width of the plate
- $r \rightarrow$ array of circuits widths, ordered by decreasing circuits areas
- $d \rightarrow$ array of circuits heights, ordered by decreasing circuits areas
- $n \rightarrow$ total number of circuits

For modeling purposes, starting from instances data, other constant quantities are defined:

- $h_{max} \rightarrow$ maximum plate height, computed as explained in Common preprocessing section.
- $h_{min} \rightarrow$ minimum plate height, computed as

$$h_{min} = \frac{\sum_{i=1}^{n} r_i \cdot d_i}{w}$$

  Indeed, the denominator is nothing but the minimum total area, namely the sum of all circuits areas. Supposing to be able to place all circuits such that the actual total area is equal to the minimum one, the height of the plate would simply be the area's value over the width of the plate, which is exactly h_{min}.

Variables

The SMT formalization of both models is based on the definition of two arrays of variables of size n:

- $x \rightarrow$ bottom left x coordinates of circuits
- $y \rightarrow$ bottom left y coordinates of circuits

and the single variable

- $height \rightarrow$ height of the plate, to be minimized, expressed as a function of y variables:

$$height = \max_{i \in N}(d_i + y_i)$$

However, the rotation model needs the addition of three specific arrays of n variables, in order to be able to express rotation constraints:

- $dx \rightarrow$ actual widths of circuits
- $dy \rightarrow$ actual height of circuits
- $rot \rightarrow$ array of boolean variables, where each of them takes value 1 if in the solution the correspondent circuit is rotated with respect its original dimensions, 0 otherwise

Constraints

No rotation model

Firstly, it was necessary to specify that position coordinates variables must be greater or equal 0, by defining

$$\forall i \in \{1..n\} \quad x_i \geq 0 \tag{14}$$
$$\forall i \in \{1..n\} \quad y_i \geq 0 \tag{15}$$

In order to reduce the search space by limiting the domain of the *height* decision variable, the following constraint is introduced:

$$h_{min} \leq height \leq h_{max} \tag{16}$$

where h_{min} and h_{max} are computed as explained in Parameters section. It is important to notice that, since the *height* definition is linked to y variables, (16) indirectly acts on y upper and lower bounds.

Similarly, an upper bound for x was not explicitly introduced, as it was already implied by

$$\max_{i \in \{1..n\}} (r_i + x_i) \leq w \tag{17}$$

which ensures that circuits do not exceed the plate's boundaries.

Moreover, the no-overlapping relationship between each pair of circuits must be imposed, which in SMT can be expressed by the disjunction of the following constraints:

$$\forall_{i,j \in \{1..n\} | i \neq j} :$$

$$x_i + r_i \leq x_j$$
$$y_i + d_i \leq y_j$$
$$x_j + r_j \leq x_i$$
$$y_j + d_j \leq y_i$$

More precisely, the i_{th} and the j_{th} circuits do not overlap if at least one of those constraints is satisfied. Formally, it is obtained:

$$\forall_{i,j \in \{1..n\} | i \neq j} \quad x_i + r_i \leq x_j \vee y_i + d_i \leq y_j \vee x_j + r_j \leq x_i \vee y_j + d_j \leq y_i \tag{18}$$

As already done for the CP model, in order to optimize as much as possible the plate available space, the intuition about the parallelism of the VLSI problem with the scheduling was exploited. The idea was to write an SMT implementation of the Minizinc `cumulative` global constraint, which can be efficiently used to manage with scheduling problems.
In particular, in order to correctly apply the constraint, it was necessary to interpret the VLSI problem as it was a cumulative resource usage problem, establishing a correspondence between the actual and the formal sets of variables:

- circuits represent the tasks to be scheduled
- y coordinates positions represent the circuits starting times
- d array of circuits heights stands for the durations of tasks
- r array of circuits widths stands for the resource requirements of tasks
- w represents the global resource bound not to exceed at any one time

Thus, the `cumulative` constraint can be expressed as:

$$\sum_{i \in \{1..n\} | y_i \leq u \wedge u < y_i + d_i} r_i \leq w \qquad \forall u \in d \tag{19}$$

Rotation model

Regarding the rotation model, it exploits all the constraints already introduced in the previous section for the no-rotation one. The only difference is that width and height of single circuits are, in this case, modelled by dx and dy variables respectively, as to allow a possible rotation of circuits, handled as a swap between the original dimensions. Formally, it is obtained:

$$\forall i \in \{1..n\} \quad x_i \geq 0 \tag{20}$$
$$\forall i \in \{1..n\} \quad y_i \geq 0 \tag{21}$$
$$h_{min} \leq height \leq h_{max} \tag{22}$$

$$\max_{i \in \{1..n\}} (dx_i + x_i) \leq w \tag{23}$$

$$\forall_{i,j \in \{1..n\} | i \neq j} \quad dx_i + r_i \leq x_j \vee y_i + dy_i \leq y_j \vee x_j + dx_j \leq x_i \vee y_j + dy_j \leq y_i \tag{24}$$

$$\sum_{i \in \{1..n\} | y_i \leq u \wedge u < y_i + dy_i} dx_i \leq w \qquad \forall u \in dy \tag{25}$$

Then, other specific constraints have been added. In particular

$$\forall i \in \{1..n\} :$$

$$\begin{gathered} dy_i = if(rot_i, r_i, d_i) \\ \wedge \\ dx_i = if(rot_i, d_i, r_i) \end{gathered} \tag{26}$$

actually allows circuits to rotate. More in details, each circuit is associated with a different *rot* boolean variable, assigned to `true` if in the solution the correspondent circuit has been rotated with respect the original dimensions, to `false` otherwise.
In particular, if the solution sets rot_i=`true`, according to (26) the i_{th} circuit width dx_i is exactly equal to d_i and the i_{th} circuit height dy_i is exactly equal to r_i. Thus, when rot_i is `true`, width and height of the i_{th} circuit turn out to be swapped, meaning that in the solution it underwent a rotation.

Moreover, it turned out that forcing square circuits not to rotate, by imposing correspondent *rot* variables to be `false`, actually improves the performances, as it reduces the search space on the *rot* decision variables. Formally, it is obtained:

$$\forall_{i \in \{1..n\}} \quad d_i = r_i \implies rot_i = false \tag{27}$$

Since it contains a larger number of variables and constraints, the rotation model results to be more complex with respect the no-rotation one. Thus, the idea was to implement an SMT version of the Minizinc lexicographic order global constraint, with the aim of ruling out all the symmetric variants of a single solution with respect the vertical axis, and so reducing the solutions space:

$$\forall_{i \in \{1..n\}} \quad lex(x_i, w - x_i - dx_i) \tag{28}$$

In particular, (28) symmetry breaking constraint allows to exclude all those solutions where x and dx take values that make $w - x - dx$ elements lexicographically smaller then x.
It is important to underline that (28) lets the rotation model to show better performances, while it significantly worsen the results of the no-rotation one. This is why the use of symmetry breaking constraint is not shared between the two models.

Objective function

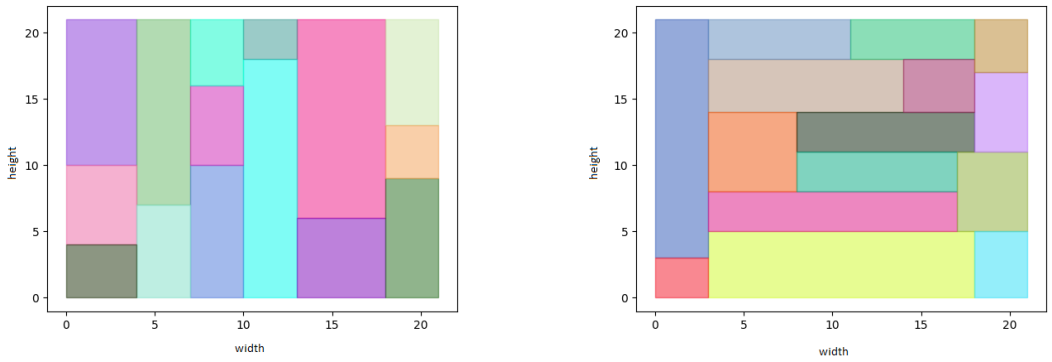
According to VLSI problem, the objective function is focused on the plate *height* minimization, as the SMT solver must be able to allocate all circuits on the plate such that the resulting height is as small as possible, also satisfying all defined constraints. Thus, the plate *height* is modelled as a decision variable and it's linked by definition to the y coordinates of circuits, which are also indirectly influenced by the minimization.
However, thanks to (16) and the equivalent (22), *height* domain is limited between a lower and an upper bound, namely h_{min} and h_{max}, such that the solution space is narrowed and the search can be more efficient.

Solver and model implementation

The tool used to solve the SMT problem is the Z3 solver, through the Z3 API in Python, called Z3Py.
In particular, Z3Py offers specific functions which allow to create integer, real or boolean variables and it supports boolean operators like `And, Or, Not, Implies` to be applied in constraints definition.
Moreover, Z3Py provides the class `Optimize()`, which is an API for solving problems using an objective function, thus finding a proper assignment for all variables by optimizing a specific quantity. Once the optimize solver is created, through the `add()` method constraints can be asserted and the `minimize()` function allows to specify the objective function, being in this case the *height* variable. In addition, the `set()` method can be use in order to limit the solver search by a timeout, in this case equal to 300s, after which it simply stops if it couldn't be able to find the optimal solution.

Results and performances

All the 40 instances have been tested applying both models. As expected, the rotation model turned out to be harder to solve with respect to the no-rotation one, as it is more constrained and it requires the definition of additional variables, and the performances were different between the two.
In particular, with a timeout of 300s, the no-rotation model solved a lager number of instances employing a shorter amount of time for each of them. Unlike CP and MIP problems, in SMT there was no way to obtain any sub-optimal solution when the solver wasn't able to find the optimal one within the time limit.
As an example, it is interesting to visualize the two different solutions found by the solver to the 14^{th} instance when applying the two models. Both of them lead to an optimal solution, which is however different between the two models, as it can be observed from the plots below:



(a) No-rotation model optimal solution

(b) Rotation model optimal solution

Figure 4: Comparison between different models solutions to instance 14

The barplot below shows all solving times, employing marked colors for those instances whose optimal solution was found within 300s, while shading tonality for those for which the solver ran out of time.

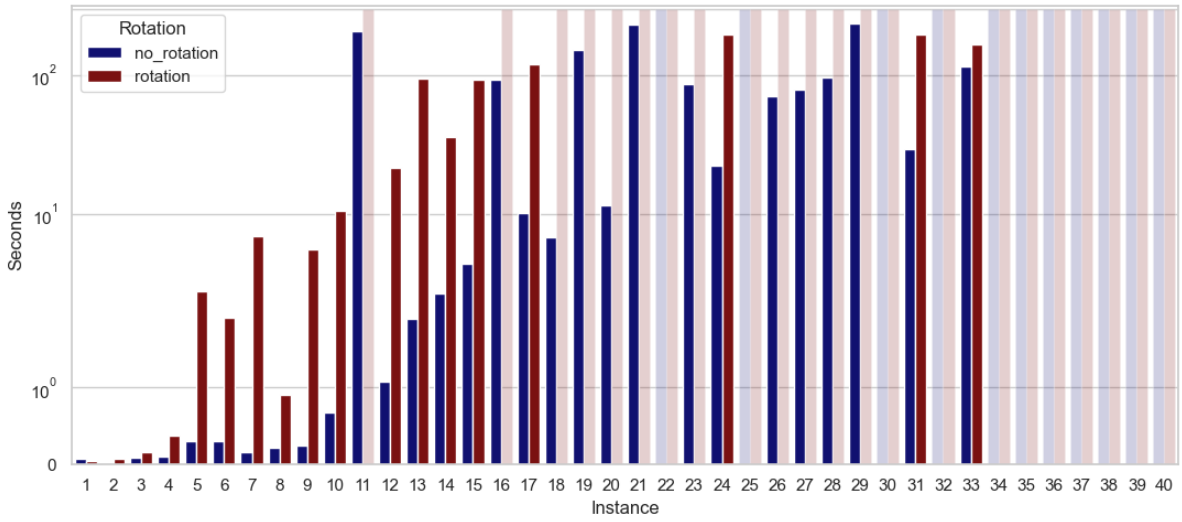


Figure 5: Barplot of SMT solving times

To better analyse solving times, all of them have been collected in the following table:

	No rotation	Rotation
Instance	Solving time(s)	Solving time (s)
1	0.071	0.044
2	0.025	0.079
3	0.089	0.166
4	0.1	0.382
5	0.31	2.773
6	0.306	1.913
7	0.154	6.939
8	0.212	0.911
9	0.243	5.529
10	0.674	10.565
11	206.614	-
12	1.075	21.448
13	1.892	94.795
14	2.637	35.842
15	4.384	93.017
16	92.245	-
17	10.099	120.406
18	6.8	-
19	152.085	-
20	11.478	-
21	231.835	-
22	-	-
23	86.57	-
24	22.369	197.017
25	-	-
26	70.266	-
27	78.214	-
28	95.472	-
29	234.153	-
30	-	-
31	29.171	197.047
32	-	-
33	116.388	165.626
34	-	-
35	-	-
36	-	-
37	-	-
38	-	-
39	-	-
40	-	-

Figure 6: Table with SMT solving times details

Finally, the addition of the symmetry breaking constraint lets the rotation model to improve its performances, but the overall results of the no-rotation model are clearly better, both in solving times and number of instances solved.

MIP model

The third optimization method employed to address the VLSI problem is the Mixed-Integer Programming (MIP), which is a linear programming technique where some of the variables involved are restricted to be integers.

No rotation model

The base version of the VLSI problem can be formulated as follows:

$$\min \quad height$$

$$\begin{aligned}
\text{s.t.} \quad & x_i + r_i \leq w && \forall i \in \{1..n\} && (29)\\
& y_i + d_i \leq height && \forall i \in \{1..n\} && (30)\\
& x_i + r_i \leq x_j + M \cdot z_{ij0} && \forall i,j \in \{1..n\} && (31)\\
& y_i + d_i \leq y_j + M \cdot z_{ij1} && \forall i,j \in \{1..n\} && (32)\\
& x_j + r_j \leq x_i + M \cdot z_{ij2} && \forall i,j \in \{1..n\} && (33)\\
& y_j + d_j \leq y_i + M \cdot z_{ij3} && \forall i,j \in \{1..n\} && (34)\\
& \sum_{k=1}^{4} z_{ijk} \leq 3 && \forall i,j \in \{1..n\} && (35)\\
& area = height \cdot w && && (36)\\
& area \leq area_{max} && && (37)\\
& area \geq area_{min} && && (38)\\
& x_i \leq w && \forall i \in \{1..n\} && (39)\\
& y_i \leq h_{max} && \forall i \in \{1..n\} && (40)\\
& x_i \in \mathbb{N} && \forall i \in \{1..n\} &&\\
& y_i \in \mathbb{N} && \forall i \in \{1..n\} &&\\
& z_{ijk} \in \{0,1\} && \forall i,j \in \{1..n\}, k \leq 4 &&\\
& height \in \mathbb{N} && &&
\end{aligned}$$

Rotation model

The rotation variant of the VLSI problem can be instead formulated as follows:

$$\min \quad height$$

$$\text{s.t.} \quad x_i + dx_i \leq w \qquad \forall i \in \{1..n\} \tag{41}$$
$$y_i + dy_i \leq height \qquad \forall i \in \{1..n\} \tag{42}$$

$$x_i + dx_i \leq x_j + M \cdot z_{ij0} \qquad \forall i, j \in \{1..n\} \tag{43}$$
$$y_i + dy_i \leq y_j + M \cdot z_{ij1} \qquad \forall i, j \in \{1..n\} \tag{44}$$
$$x_j + dx_j \leq x_i + M \cdot z_{ij2} \qquad \forall i, j \in \{1..n\} \tag{45}$$
$$y_j + dy_j \leq y_i + M \cdot z_{ij3} \qquad \forall i, j \in \{1..n\} \tag{46}$$
$$\sum_{k=1}^{4} z_{ijk} \leq 3 \qquad \forall i, j \in \{1..n\} \tag{47}$$

$$area = height \cdot w \tag{48}$$
$$area \leq area_{max} \tag{49}$$
$$area \geq area_{min} \tag{50}$$

$$x_i \leq w \qquad \forall i \in \{1..n\} \tag{51}$$
$$y_i \leq h_{max} \qquad \forall i \in \{1..n\} \tag{52}$$

$$dx_i = rot_i \cdot d_i + (1 - rot_i) \cdot r_i \qquad \forall i \in \{1..n\} \tag{53}$$
$$dy_i = rot_i \cdot r_i + (1 - rot_i) \cdot d_i \qquad \forall i \in \{1..n\} \tag{54}$$

$$x_i \in \mathbb{N} \qquad \forall i \in \{1..n\}$$
$$y_i \in \mathbb{N} \qquad \forall i \in \{1..n\}$$
$$z_{ijk} \in \{0, 1\} \qquad \forall i, j \in \{1..n\}, k \leq 4$$
$$height \in \mathbb{N}$$
$$dx_i \in \mathbb{N} \qquad \forall i \in \{1..n\}$$
$$dy_i \in \mathbb{N} \qquad \forall i \in \{1..n\}$$
$$rot_i \in \{0, 1\} \qquad \forall i \in \{1..n\}$$

Parameters

Regarding the constant quantities involved in the model, the notation used is as follows:

- $w \rightarrow$ width of the plate
- $r_i \rightarrow$ original width of the i_{th} circuit
- $d_i \rightarrow$ original height of the i_{th} circuit
- $n \rightarrow$ total number of circuits
- $h_{max} \rightarrow$ maximum plate height, computed as explained in Common preprocessing section.
- $area_{max} \rightarrow$ maximum area of the whole plate, computed as

$$area_{max} = h_{max} \cdot w$$

- $area_{min} \rightarrow$ minimum area of the plate, computed as

$$area_{min} = \sum_{i=1}^{n} r_i \cdot d_i$$

- $M \rightarrow$ large enough constant, possibly equal to h_{max}, used to express Big-M constraints

Variables

The way a problem is modelled firstly concerns the choice of the decision variables and how they are defined. In this case, the involved variables are:

- $x_i \rightarrow$ bottom left x coordinate of the i_{th} circuit
- $y_i \rightarrow$ bottom left y coordinate of the i_{th} circuit
- $z_{ijk} \rightarrow$ artificial logic variable aimed to express Big-M constraints between the i_{th} and the j_{th} circuits
- $height \rightarrow$ height of the plate, to be minimized

Variables just presented are employed regardless whether in the model the rotation is allowed or not. However, for the rotation model only, it is necessary to add some specific variables in order to be able to express rotation constraints:

- $dx_i \rightarrow$ actual width of the i_{th} circuit
- $dy_i \rightarrow$ actual height of the i_{th} circuit
- $rot_i \rightarrow 1$ if in the solution the i_{th} circuit is rotated with respect its original dimensions, 0 otherwise

It is important to notice that in the no-rotation model the width and the height of each n_{th} circuit are represented by r_n and d_n respectively, which are constant, meaning that the circuit dimensions are fixed and that any rotation can be applied.

Constraints

No rotation model

The constraints (29)(30) keep away the circuits from exceeding the plate's boundaries, reducing as a consequence the domain of its position coordinates.

Constraints from (31) to (35) aimed to avoid circuits from overlapping by using the Big-M constraints technique. More precisely, the i_{th} and the j_{th} circuits do not overlap if at least one of the following constraints is satisfied:

$$x_i + r_i \leq x_j$$
$$y_i + d_i \leq y_j$$
$$x_j + r_j \leq x_i$$
$$y_j + d_j \leq y_i$$

In MIP the disjunction between constraints can be implemented through the Big-M constraint technique. In particular, each of those 4 constraint is associated with one different logic variable z_{ijk}, which is combined with a large positive constant M, usually defined to be "large enough". The aim of the additional term $M \cdot z_{ijk}$ is to relax those constraints whose z_{ijk} variable takes the value 1 in the solution.

Thus, for each combination of i and j, the sum over k of z_{ijk} variables must be less or equal 3, meaning that at least one of the 4 no-overlap constraints involving the i_{th} and j_{th} circuits must be satisfied even without any relaxation (because the correspondent z_{ijk} is set to 0 in the solution).

Constraints from (36) to (38) are meant to furtherly restrict the solution space, by bounding the *area* between $area_{max}$ and $area_{min}$, which are computed as explained above. More precisely, the actual *area* must be greater or equal the minimum area, which is nothing but the sum of all circuits areas, and it must be lower or equal the maximum area, computed as the product between the plate's width and h_{max}. This implicitly means that $height \leq h_{max}$ must hold.

The constraints (39) and (40) simply remark the domain reduction on x and y variables, which obviously cannot be over the plate width or higher the maximum possible height.

Rotation model

Regarding the rotation model, constraints from (41) to (52) have exactly the same meaning of constraints from (29) to (40) of the no-rotation one already discussed above, with the only difference that width and height of single circuits are, in this case, modelled by dx and dy variables respectively. Indeed, the rotation of a circuit is handled as a swap between its width and height, making necessary the modelling of circuits dimensions as decision variables.

Constraints (53) and (54) actually allow circuits to rotate by associating with each of them one different *rot* logic variable, which takes the value 1 if in the solution the correspondent circuit has been rotated with respect its original dimensions, 0 otherwise.
In particular, if the solution sets $rot_i = 1$, according to (53) the width of the i_{th} circuit dx_i is exactly equal to d_i, being the contribution of r_i multiplied by $(1 - rot_i)$, which results to be 0. Then, according to (54), the i_{th} circuit height dy_i is exactly equal to r_i for the very same reason. Thus, when rot_i is 1, width and height of the i_{th} circuit turn out to be swapped, meaning that in the solution it underwent a rotation.

Objective function

According to the VLSI problem, the quantity needed to be optimize is the *height* of the plate. Thus, it has been modelled as a decision variable and the objective function is focused on its minimization.
Several constraints involving *height*, such as (30) and the equivalent (42), link it to y coordinates of circuits, which as a consequence implicitly feel the effect of the minimization. Moreover, constraints from (36) to (38) and equivalents indirectly set a lower and an upper bound for *height* value, furtherly narrowing its domain.

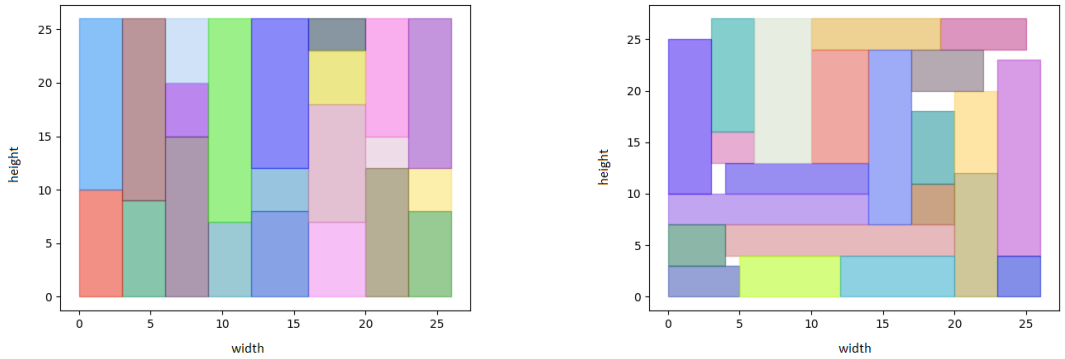
Solver and model implementation

The solver used to approach the MIP problem is Gurobi Optimizer, run through its Python interface by using the `gurobipy` module.
In particular, `gurobipy` module offers specific methods which allow to create the desired model, adding variables and defining related constraints, then setting through `setObjective()` method the optimization objective, stating if it has to be minimized or maximized. Finally, the model thus built can be optimized by applying the `optimize()` method.
In this case, the `setParam()` method was used to set a time limit for the optimization, in particular equal to 300s, after which the solver simply stops the search and returns the last best solution found.

Results and performances

For both models, all the 40 instances have been tested. As expected, performances were different between the rotation and the no-rotation model, as allowing the rotation of circuits means to add specific constraints and to have more decision variables, increasing the computational complexity.
With a timeout of 300s, the solver found at least one solution for all the instances, even if for the hardest ones it could not be able to find the optimal one.

As a matter of example, it is interesting to visualize the two different solutions found by the solver to the 19^{th} instance when applying the two models. It's important to notice that the no-rotation model leads to an optimal solution, while the rotation one only provides a sub-optimal solution, as it can be observed from the two plots below:



(a) No-rotation model optimal solution

(b) Rotation model sub-optimal solution

Figure 7: Comparison between different models solutions to instance 19

Solving times were then collected in an intuitive barplot, which shows marked colors for those instances whose optimal solution was found within 300s, while shading tonality for those for which the solver ran out of time.

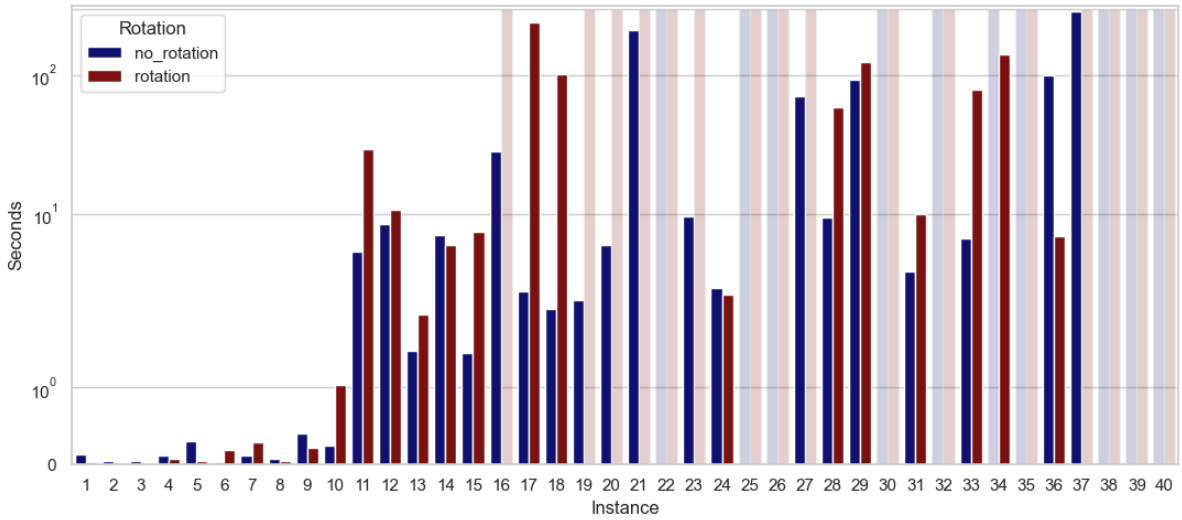


Figure 8: Barplot of MIP solving times

Solving times can be seen in details in the following table:

	No rotation	Rotation
Instance	Solving time(s)	Solving time (s)
1	0.132	0.027
2	0.051	0.01
3	0.052	0.008
4	0.122	0.081
5	0.298	0.052
6	0.028	0.19
7	0.116	0.29
8	0.07	0.041
9	0.401	0.217
10	0.252	1.04
11	5.303	29.348
12	8.416	10.617
13	1.471	1.946
14	6.96	5.92
15	1.452	7.405
16	28.122	-
17	2.765	240.089
18	2.037	101.227
19	2.368	-
20	5.959	-
21	211.037	-
22	-	-
23	9.614	-
24	2.925	2.623
25	-	-
26	-	-
27	69.8	-
28	9.422	58.048
29	92.475	124.418
30	-	-
31	3.841	9.867
32	-	-
33	6.572	78.343
34	-	141.746
35	-	-
36	98.829	6.864
37	286.035	-
38	-	-
39	-	-
40	-	-

Figure 9: Table with MIP solving times details

Finally, the no-rotation model clearly showed better performances, as it allows to solve a larger number of instances and, in most of the cases, to solve them in a shorter time.

Conclusions

The complexity of the VLSI problem and its versions approached made necessary a fine refinement and tuning of the employed models. As a consequence, lots of experimental running has been analysed and progressively improved, trying to reach reasonable performances.
Eventually, all three optimization methods used to approach the problem showed satisfactory results. However, CP technique turned out to be way more efficient than SMT and MIP ones, allowing to optimally solve 39 out of 40 instances for the no-rotation version of the problem and 34 out of 40 instances for the rotation one, employing overall better solving times.

References

[1] Eric Huang, Richard E. Korf *Optimal Rectangle Packing: An Absolute Placement Approach* , 2012. `https://arxiv.org/ftp/arxiv/papers/1402/1402.0557.pdf`.

[2] Stephan Held, Bernhard Korte, Dieter Rautenbach, and Jens Vygen *Combinatorial Optimization. in VLSI Design* `https://www.or.uni-bonn.de/research/montreal.pdf`.

[3] Hay-Wai Chan and Igor L. Markov *Symmetries in Rectangular Block-Packing* `https://web.eecs.umich.edu/~imarkov/pubs/misc/symcon03-fp.pdf`.

[4] *The Minizinc Handbook* `https://www.minizinc.org/doc-2.5.3/en/index.html`.

[5] *Z3 API in Python* `https://ericpony.github.io/z3py-tutorial/guide-examples.htm`.

[6] *Z3 Namespace Reference* `https://z3prover.github.io/api/html/namespacez3py.html`.

[7] *Gurobi Optimization, Python API Details* `https://www.gurobi.com/documentation/9.5/refman/py_python_api_details.html`.