Project work in Combinatorial Decision Making and Optimization

Very Large Scale Integration - VLSI Design

Stefano Poggi Cavalletti
stefano.poggi2@studio.unibo.it

Table of Contents

1 Introduction

VLSI (Very Large Scale Integration) refers to the trend of integrating circuits into silicon chips. Given a fixed-width plate and a list of rectangular circuits, the problem consists in deciding how to place the circuits on the plate in order to minimize the length of the final device.

The combinatorial optimization problem will be modeled and solved using two approaches: Constraint Programming (CP) using MiniZinc and Satisfiability Modulo Theories (SMT) using Z3Py. In both cases, in addition to a base model, it will be developed a model which takes into account the rotation of the circuits, in order to see which modifications should be applied. Finally, the obtained results will be compared to assess the performances of the solvers using different models.

The **input** consists of a set of 40 instances in a text format, each one containing the following data:

- width of the silicon plate w
- number of circuits to place n
- horizontal dimension x_i of the i-th circuit, $\forall i \in \{1..n\}$
- vertical dimension y_i of the i-th circuit, $\forall i \in \{1..n\}$

The output, instead, will be structured in the following way:

$w \quad h$
n
$x_0 \quad y_0 \quad \hat{x_0} \quad \hat{y_0}$
$x_1 \quad y_0 \quad \hat{x_1} \quad \hat{y_1}$
...
$x_n \quad y_n \quad \hat{x_n} \quad \hat{y_n}$

where w, n, x_i and y_i are the same as previously described; h is the maximum height reached by the circuits configuration; $\hat{x_i}$ and $\hat{y_i}$ are the coordinates of the left-bottom corner of the i-th circuit, $\forall i \in 1..n$. The objective variable is the final plate's *height* and the goal is to minimize it.

Therefore, in the various models that will be described, the variables used to formalize the problem are the following:

- $width$ is the fixed plate width
- n is the number of circuits to place in the plate
- x_dim and y_dim are two arrays (indexed from 1..n) representing the horizontal and vertical dimensions of the circuits, respectively
- x and y are two arrays (indexed from 1..n) representing the horizontal and vertical coordinates, respectively, of the bottom-left coordinate of the circuits

2 CP

The first technique used to approach the problem was modeling with CP (Constraint Programming) using Minizinc.

For clarification purposes, we report the structure of each input instance encoded in a *.dzn* file, which is readable by MiniZinc:

$w = width$
$n = number\ of\ circuits$
$x\_dim = [x_0,\ x_1,\ ...,\ x_n]$
$y\_dim = [y_0,\ y_1,\ ...,\ y_n]$

2.1 Variables

First of all, all the parameters, decision variables and objective variables of the problem had to be defined.

The **input and output parameters** were the ones previously described in the introduction, which were defined in the model using the MiniZinc syntax.

We want to model our variables to have the smallest possible domain in order to reduce the search space. Hence, we reduced the variable domain to make the model more efficient. This was done by defining a range for each variable of the array x: the bottom left corner of each rectangle i cannot be placed farther along the x axis than the plate width minus its horizontal dimension, otherwise it would fall outside the plate. A similar reasoning can be defined for the domain of the variables y_i: the bottom left corner of each rectangle i cannot be placed higher on the y axis than the height of the plate minus its vertical dimension.

$$\forall i \in \{1..n\} \quad x_i \leq width - x\_dim_i$$

$$\forall i \in \{1..n\} \quad y_i \leq height - y\_dim_i$$

Moreover, with the same purpose of reducing variables domain, upper and lower bounds were defined for the plate's height:

- lower bound: the plate's height cannot be smaller than the height of the tallest rectangle
$$lowb = max(y\_dim_i) \quad \forall i \in \{1..n\}$$
- upper bound: the plate's height must be below the sum of all the rectangles' heights
$$upb = sum(y\_dim_i) \quad \forall i \in \{1..n\}$$

2.2 Objective function

The **objective function** is to minimize the objective variable *height*, which is the height of the plate.

It ranges from the lower bound *lowb* to the upper bound *upb* which were previously described and it is defined as:

$$height = max(y_i + y\_dim_i \mid i \in \{1..n\})$$

2.3 Constraints

One of the most important parts of the problem is related to the definition of different constraints and their propagation in order to remove inconsistent values from variables domain.

The use of **global constraints** is useful for the solution of this problem, since it allows to obtain more efficient solutions thanks to propagation algorithms. Two different global constraints were used: **cumulative** and **diffn**.

The *cumulative* constraint was taken from the modeling of scheduling problems: it is usually used when we want to constrain the usage of shared resources by different tasks. In general, it 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.

Our problem can be seen as one of this kind. More specifically, for the x axis, the plate's height represents the resource bound and each circuit is a task where its x coordinate represents the starting time, its width represents the duration, while the circuit's height represents the required resource. The same reasoning can be applied symmetrically for the y axis.

```
cumulative(x, x_dim, y_dim, height)
cumulative(y, y_dim, x_dim, width)
```

The *diffn* constraint is a non-overlapping constraint which, given the vectors of circuits' bottom-left coordinates x and y and the vectors of their dimensions x_dim and y_dim, it prevents the circuits from overlapping:

```
diffn(x, y, x_dim, y_dim)
```

Furthermore, **symmetry braking constraints** were applied to reduce the number of solutions: the solver may in fact explore many symmetric variants of the same solution. The basic idea behind symmetry breaking is to impose an order. In our case it was decided to place always the biggest circuit in the bottom left part of the plate. The rationale behind this choice is that, by taking up the most space on the plate, it is the most likely circuit to have an impact on the positioning of subsequent circuits.

To do this we need to define an order of the circuits, which is done by sorting them in descending order considering their area, given by the product between x_dim and y_dim. We call `ord_circ` the array of sorted circuits by their area.

After that, we force the x and y coordinates of the bottom-left corner of the biggest circuit to be 0, meaning that it will be placed in the bottom-left part of the plate:

$$\texttt{x[ord\_circ[1]] = 0} \wedge \texttt{y[ord\_circ[1]] = 0}$$

2.4 Rotation model

If we decide to take into account the rotation of the circuits, we need to perform some model modifications. This is done by introducing a boolean array `rotation` indexed by the number of circuits which specifies if each circuit is rotated or not.

In case of rotation each rectangle will have its width and height values swapped, therefore we need to update their actual values of widths and heights accordingly:

$$\forall i \in \{1..n\} \quad x\_dim\_rot_i = \begin{cases} y\_dim_i & \text{if rotation}_i \\ x\_dim_i & \text{otherwise} \end{cases}$$

$$\forall i \in \{1..n\} \quad y\_dim\_rot_i = \begin{cases} x\_dim_i & \text{if rotation}_i \\ y\_dim_i & \text{otherwise} \end{cases}$$

As a consequence, the objective variable *height* and the constraints seen before are modified by simply replacing the original dimension variables with x_dim_rot and y_dim_rot to allow rotation.

We also introduce a new constraint, which says that a circuit cannot be rotated if its height is greater than the plate's width:

$$\forall i \in \{1..n\} \quad y\_dim_i > width \implies rotation_i = false$$

Moreover, another symmetry braking constraint is added due to the fact that, in case of rectangles which are squares (meaning that they have the same dimensions), we avoid the rotation forcing width and height to be the original ones.

$$\forall i \in \{1..n\} \quad x\_dim_i = y\_dim_i \implies (x\_dim\_rot_i = x\_dim_i \wedge y\_dim\_rot_i = y\_dim_i)$$

The output file is also modified so that the string *"rotated"* is printed next to the coordinates of the output circuits: this indicates that the related circuit has been rotated.

As an example, a possible output for the instance 3 is the following:

10 10
6
3 3 4 7
3 4 0 7 *rotated*
3 6 7 4
3 7 0 4 *rotated*
4 4 6 0
4 6 0 0 *rotated*

2.5 Hardware setup

The experiments were performed on a machine with the following specifications:

- Apple M1 Pro 8-core
- 16 GB of RAM
- macOS 13.0

The project was developed using Python 3.10.6 and MiniZinc 2.6.4.

2.6 Validation

The model was implemented with MiniZinc and run using Gecode solver. Different combinations of search heuristics, for variables and domains, as well as restart strategies were employed to compare the model performances. In particular, as variables heuristics we considered:

- *input_order*, that chooses the variable in order from the array
- *first_fail*, that chooses the variable with the smallest domain size
- *dom_w_deg*, that chooses the variable with the smallest value of domain size divided by weighted degree, which is the number of times it has been in a constraint that caused failure earlier in the search

As domain heuristic, the choice was made on *indomain_min*, that assigns the variable with its smallest domain value.

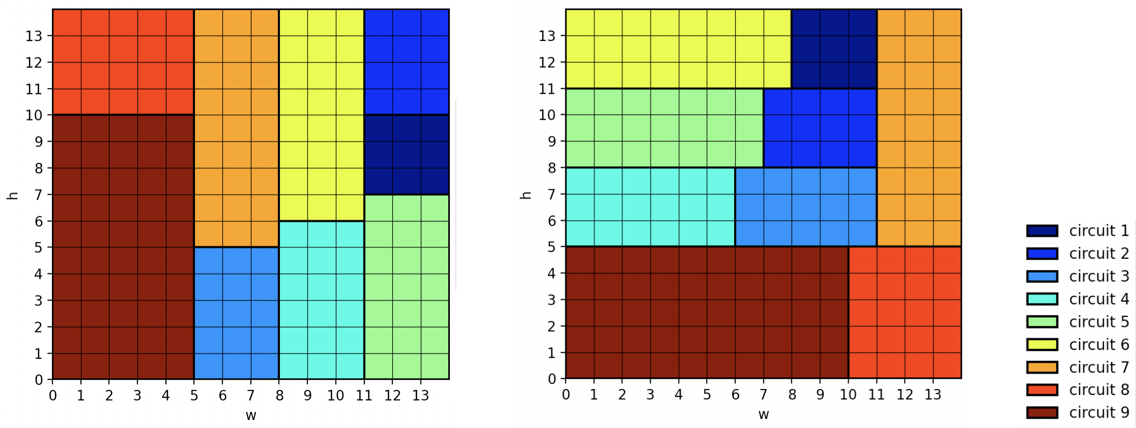
Also, restarting the search is useful to introduce randomness and break deterministic behaviour in searching solutions. In the model, different restart strategies were chosen:

- restart_constant(100)
- restart_linear(100)
- restart_geometric(1.5, 100)
- restart_luby(100)

The results compare the performances of the base model (no rotation) and of the rotation model, both tested with and without symmetry braking constraints.

During the experiments, a time limit of 300 seconds was imposed: if the solver was not able to find a solution within the time limit, the solving process was aborted.

After experimenting with different combinations, the search strategy configuration which produced the best trade-off between the number of solved instances and solving time was *input_order* and *indomain_min* combined with *geometric restart*. The results are shown in Table 1.



(a) Solution without rotation

(b) Solution with rotation

(c) Legend of circuits

Figure 1: Example of a possible solution for the instance n. 7

Table 1: Solving times and height obtained by CP models (base and rotation) with and without symmetry braking constraints

Instance	Base + SB	Base w/o SB	Rot + SB	Rot w/o SB	Height
1	0,194	0,287	0,268	0,217	8
2	0,192	0,202	0,219	0,219	9
3	0,196	0,205	0,206	0,217	10
4	0,192	0,207	0,246	0,312	11
5	0,212	0,208	1,354	0,255	12
6	0,200	0,213	1,165	1,499	13
7	0,228	0,206	3,688	46,06	14
8	0,235	0,235	0,225	0,212	15
9	0,198	0,207	19,26	87,64	16
10	0,199	0,210	-	-	17
11	0,203	0,206	-	-	18
12	0,228	0,218	0,523	0,224	19
13	0,202	0,204	-	-	20
14	0,222	0,231	-	-	21
15	0,193	0,250	0,266	0,227	22
16	1,317	-	-	-	23
17	6,366	6,158	0,236	0,215	24
18	1,434	1,436	1,120	-	25
19	-	-	-	-	-
20	5,908	5,771	-	-	27
21	0,227	0,211	-	-	28
22	-	-	-	-	-
23	0,266	-	-	-	30
24	0,231	20,68	0,228	0,309	31
25	-	-	-	-	-
26	147,4	-	0,918	-	33
27	0,233	60,50	0,221	0,286	34
28	17,23	-	-	-	35

Continued on next page

Table 1 – continued from previous page

Instance	Base w/ SB	Base w/o SB	Rot w/ SB	Rot w/o SB	Height
29	0,246	-	-	-	36
30	-	-	-	-	-
31	-	-	271,7	-	38
32	-	-	-	-	-
33	0,231	0,309	0,242	0,218	40
34	-	-	-	-	-
35	-	-	-	-	-
36	0,232	0,278	-	-	40
37	-	-	-	-	-
38	-	-	-	-	-
39	-	-	-	-	-
40	-	-	-	-	-

3 SMT

Another approach to this problem involves SMT (Satisfiability Modulo Theories) with Z3Py, that was chosen because of its expressive power (due to the use of first-order logic) compared to SAT, which instead can be complex to formalize. The main drawback of SMT models is the worse efficiency, which however is compensated by the higher expressivity.

3.1 Variables

Modeling with SMT followed the same scheme that was seen for CP. First, the coordinates of the circuits have been encoded in two vectors x and y of integers with dimension given by the number of circuits. As previously seen, the arrays x_dim and y_dim contain the widths and heights of all the circuits, while *height* represents the maximum height of the plate.

As done before, the variable domain has been reduced to speed up the search: first of all, the horizontal coordinates x_i have to be greater than zero and their sum with the circuit widths must be below the plate width.

$$\forall i \in \{1..n\} \quad x_i \geq 0 \ \wedge x_i + x\_dim_i \leq w$$

Similarly, the vertical coordinates y_i have to be greater than zero and their sum with the circuit heights must be below the maximum plate height.

$$\forall i \in \{1..n\} \quad y_i \geq 0 \ \wedge y_i + y\_dim_i \leq height$$

3.2 Objective function

The **objective function** and its bounds are the ones which were previously described in section 2.2.

3.3 Constraints

The main constraints are meant to tell the solver to respect the boundaries of the plate for the horizontal and vertical coordinates:

$$\forall i \in \{1..n\} \quad x_i + x\_dim_i \leq w$$
$$\forall i \in \{1..n\} \quad y_i + y\_dim_i \leq height$$

Furthermore, a similar definition of the global constraint *cumulative* (previously used in Minizinc for the CP solution) was given, with the same reasoning that was described before.

This is defined in the following way:

$$cumulative(y, y\_dim, x\_dim, width)$$
$$cumulative(x, x\_dim, y\_dim, height)$$

Moreover, a non-overlapping constraint is necessary to make sure that no pair of rectangles overlap:

$$\forall i, j \in \{1..n\}, i \neq j \quad (x_i + x\_dim_i \leq x_j) \vee (x_j + x\_dim_j \leq x_i) \vee (y_i + y\_dim_i \leq y_j) \vee (y_i + y_j + y\_dim_j \leq y_i)$$

Symmetry braking constraints are the same as in CP: it has been decided to put the biggest circuit in the bottom left part of the plate, at x and y coordinates equal to zero.

3.4 Rotation model

As seen in CP, for the rotation model we introduce two other vectors x_dim_rot and y_dim_rot which contain the actual widths and heights considered, since in case of rotation the values of width and height will be swapped.

$$\forall i \in \{1..n\} \quad (x\_dim\_rot_i = x\_dim_i \ \wedge \ y\_dim\_rot_i = y\_dim_i) \vee \\ \vee \ (x\_dim\_rot_i = y\_dim_i \wedge y\_dim\_rot_i = x\_dim_i)$$

As previously seen, we add a constraint for those cases where a circuit has the same horizontal and vertical dimensions, hence forcing it to avoid rotation:

$$\forall i \in \{1..n\} \quad x\_dim_i = y\_dim_i \implies (x\_dim\_rot_i = x\_dim_i \wedge y\_dim\_rot_i = y\_dim_i)$$

As previously done, the output file is modified to print the string *"rotated"* in case a circuit is rotated.

3.5 Validation

The standard search method provided by Z3Py was used to compare the performance of the models: the class `Optimize` allows to get an assignment for each variable which minimizes the objective function through the method `Optimize.minimize(<minimization objective>)`, whose aim is to minimize the objective variable *height*.

Table 2: Solving times and height obtained by SMT models (base and rotation) with and without symmetry braking constraints

Instance	Base + SB	Base w/o SB	Rot + SB	Rot w/o SB	Height
1	0,005	0,025	0,017	0,032	8
2	0,010	0,088	0,048	0,065	9
3	0,014	0,010	0,084	0,124	10
4	0,024	0,039	0,286	0,631	11
5	0,040	0,073	0,782	0,709	12
6	0,061	0,129	0,539	2,751	13
7	0,078	0,113	0,808	2,567	14
8	0,091	1,127	1,282	0,515	15
9	0,102	0,179	1,100	3,558	16
					Continued on next page

Table 2 – continued from previous page

Instance	Base w/ SB	Base w/o SB	Rot w/ SB	Rot w/o SB	Height
10	0,321	0,417	44,76	137,9	17
11	-	-	-	-	-
12	0,982	1,443	71,52	26,48	19
13	0,916	1,809	41,35	122,9	20
14	2,814	3,896	107,4	-	21
15	1,166	2,115	65,36	243,9	22
16	-	-	-	-	-
17	5,682	7,712	226,4	112,9	24
18	5,793	8,449	-	-	25
19	-	-	-	-	-
20	145,0	-	-	-	27
21	-	-	-	-	-
22	-	-	-	-	-
23	16,75	23,65	-	-	30
24	12,27	11,38	-	-	31
25	-	-	-	-	-
26	94,15	117,6	-	-	33
27	21,21	21,56	-	-	34
28	33,05	40,65	-	-	35
29	60,46	67,93	-	-	36
30	-	-	-	-	-
31	7,358	11,84	238,7	-	38
32	-	-	-	-	-
33	18,65	17,31	291,4	272,0	40
34	-	-	-	-	-
35	-	-	-	-	-
36	-	-	-	-	-
37	-	-	-	-	-
38	-	-	-	-	-
39	-	-	-	-	-
40	-	-	-	-	-

4 Conclusion

VLSI design turned out to be a problem of non-negligible complexity: the goal was to find a good model through a combination of constraints and types of search in order to get the most efficient solution with as smaller solving times as possible.
In both cases with CP and SMT we were able to get satisfactory results, in particular in the base model that does not allow rotation, which inevitably adds complexity. In particular, among the two techniques, CP managed to obtain the best results by solving a larger number of instances. In all the cases, as the number of circuits in the plate increases, the complexity raises and the solver may not be able to find a solution within the time limit of 300 seconds.

To sum up, in Table 3 we report a table with the final plate's height found for each instance by each technology (CP and SMT) and the total number of solved instances:

Table 3: Height found by CP and SMT for each instance and total number of solved instances

Instance	**CP**	**SMT**
1	8	8
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	-	-
12	19	19
13	20	20
14	21	21
15	22	22
16	23	-
17	24	24
18	25	25
19	-	-
20	27	27
21	28	-
22	-	-
23	30	30
24	31	31
25	-	-
26	33	33
27	34	34
28	35	35
29	36	36
30	-	-

Continued on next page

Table 3 – continued from previous page

Instance	CP	SMT
31	38	38
32	-	-
33	40	40
34	-	-
35	-	-
36	40	-
37..40	-	-
Total (base)	28	25
Total (rotation)	18	17

5 References

[1] The MiniZinc Handbook, `https://www.minizinc.org/doc-2.3.0/en/`

[2] Using Global Constraints for Rectangle Packing, H. Simonis and B. O'Sullivan, 2008

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

[4] Programming Z3, Bjørner et al, `https://theory.stanford.edu/~nikolaj/programmingz3.html#sec-bounded-model-checking`