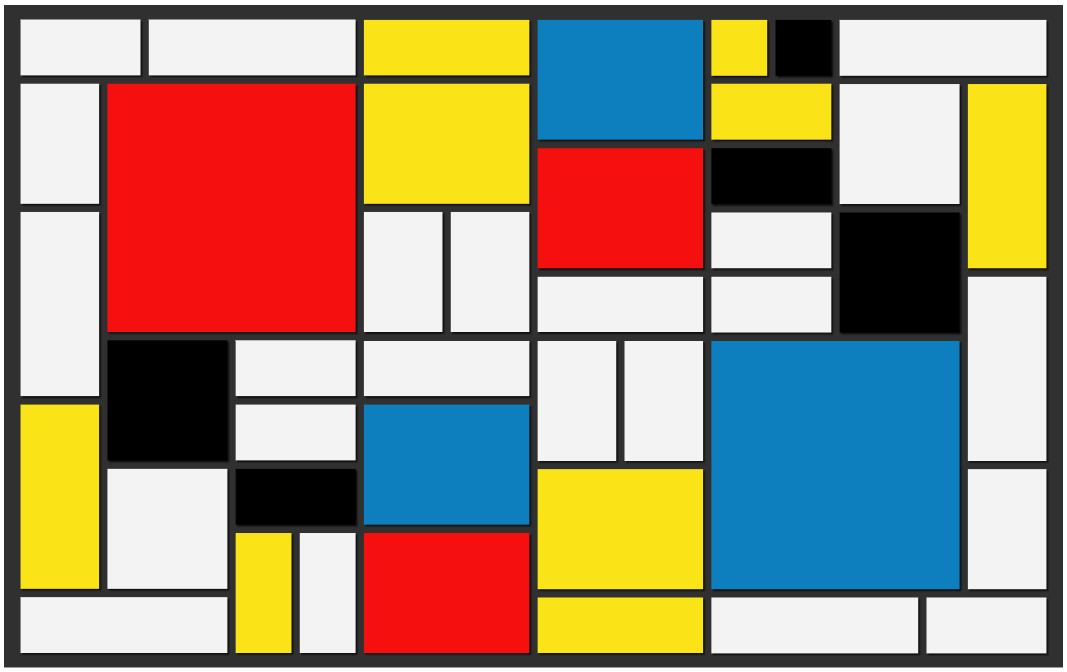
Very Large Scale Integration

Constraint Programming



Flavio Pinzarrone flavio.pinzarrone@studio.unibo.it
Enrico Pallotta enrico.pallotta@studio.unibo.it
Giuseppe Tanzi giuseppe.tanzi@studio.unibo.it

Alma Mater Studiorum - University of Bologna

Contents

1 Introduction **2**

2 Problem formulation **2**
- 2.1 Problem parameters 2
- 2.2 Decision variables 2
- 2.3 Objective function 2
- 2.4 Main problem constraints 3
  - 2.4.1 Domain Constraints 3
  - 2.4.2 No overlapping constraint 3
- 2.5 Implied Constraints 3

3 CP formulation **4**
  - 3.0.1 Implied constraints 4
  - 3.0.2 Symmetry breaking constraints 4
- 3.1 Rotation model 5
  - 3.1.1 Decision variables 5
  - 3.1.2 Main model constraints 5
  - 3.1.3 Implied Constraints 5
  - 3.1.4 Symmetry Breaking Constraints 5
- 3.2 Search strategy and solver 6

4 Results and performances **8**
- 4.1 Hardware 9

1 Introduction

VLSI (Very Large Scale Integration) refers to the trend of integrating circuits into silicon chips. A typical example is the smartphone. The modern trend of shrinking transistor sizes, allowing engineers to fit more and more transistors into the same area of silicon, has pushed the integration of more and more functions of cellphone circuitry into a single silicon die (i.e.plate).
So, given a fixed-width plate and a list of rectangular circuits, decide how to place them on the plate so that the length of the final device is minimized(improving its portability).
Consider two variants of the problem.
In the **first**, each circuit must be placed in a **fixed orientation** with respect to the others. This means that, an $n \times m$ circuit cannot be positioned as an $m \times n$ circuit in the silicon plate.
In the **second case**, the **rotation is allowed**, which means that an $n \times m$ circuit can be positioned either as it is or as $m \times n$.
The solution discussed in the following sections is based on an Constraint programming model.

2 Problem formulation

Here we define some notation that will be used in the description of the model, listing below the parameters (which can be read from the instance files) and the variables we used.

2.1 Problem parameters

- $n \in \mathbf{N}$ number of circuits to place
- $W \in \mathbf{N}$ maximum plate width
- $w_i \in \mathbf{N}$ width of the i^{th} circuit
- $h_i \in \mathbf{N}$ height of the i^{th} circuit

2.2 Decision variables

- H plate height, to be minimized
- x_i bottom left x coordinate of the i^{th} circuit
- y_i bottom left y coordinate of the i^{th} circuit

2.3 Objective function

The objective function of this problem is to minimize H. To help the CP solver, we can define the lower bound and the upper bound of H as follows:

- The lower bound for H is obtained when there aren't no empty spaces among the circuits and all the plate is fitted

$$lb = \frac{\sum_i h_i * w_i}{W}$$

- The upper bound for H is obtained where all the circuits are on a single column:

$$ub = \sum_i h_i$$

Hence:

$$lb \leq H \leq ub \tag{1}$$

2.4 Main problem constraints

In this section we provide a description of the basic constraints which need to be enforced in order to find a solution.
Basically there are two main constraints:

2.4.1 Domain Constraints

Each circuit should not exceed the plate bounds, neither in width nor in height.

$$\bigwedge_{i=1}^{n} 0 \leq x_i \leq W - w_i \tag{2}$$

$$\bigwedge_{i=1}^{n} 0 \leq y_i \leq H - h_i \tag{3}$$

2.4.2 No overlapping constraint

Circuits should not overlap with each other.

$$\bigwedge_{i,j \in [1,n] \mid i<j} (x_i + w_i \leq x_j) \lor (x_j + w_j \leq x_i) \lor (y_i + h_i \leq y_j) \lor (y_j + h_j \leq y_i) \tag{4}$$

2.5 Implied Constraints

Cumulative constraints

Since we noticed that this problem can be seen as a task scheduling problem, the following cumulative constraints can be enforced.

- **Cumulative constraint over the rows**
  The first parallelism with the scheduling problem is the following:
  - the time axis is the plate height,
  - tasks are circuits,
  - tasks duration are circuits heights,
  - tasks resource requirements are circuits widths,
  - resource capacity is the plate width;

  we enforced a cumulative constraint in our model as follows:

$$\sum_{i=1 \mid y_i \leq u < y_i + h_i}^{N} w_i \leq W \qquad u \in [1, H] \tag{5}$$

- **Cumulative constraint over the columns**
  While the second parallelism with the scheduling problem is the following:
  - the time axis is the plate width,
  - tasks are circuits,
  - task durations are circuits widths,
  - task resource requirements are circuits heights,
  - resource capacity is the plate height;

  we enforced a cumulative constraint in our model as follows:

$$\sum_{i=1 \mid x_i \leq u < x_i + w_i}^{N} h_i \leq H \qquad u \in [1, W] \tag{6}$$

3 CP formulation

In this section we provide a description of the implementation of the problem in CP formulation.

Constraints

The first two constraints (2) and (3) are easily implemented.

- **Diffn global constraint**
  To implement the `no overlapping` constraint (4) we used the `diffn` global constraint:

```
predicate diffn(array [int] of var int: x,
                array [int] of var int: y,
                array [int] of var int: dx,
                array [int] of var int: dy)
```

  It constrains rectangles i , given by their origins $(x[i], y[i])$ and sizes $(dx[i], dy[i])$, to be non-overlapping.
  Where:

  - x and y are the vectors of the left-bottom corner coordinates of the circuits
  - dx and dy are the vectors of the shapes of the circuits

  In our formulation the constraint is written as follows:
  $diffn(\{x_1, x_2, ..., x_n\}, \{y_1, y_2, ..., y_n\}, \{w_1, w_2, ..., w_n\}, \{h_1, h_2, ..., h_n\})$

3.0.1 Implied constraints

- **Cumulative global constraint**
  We implemented the `cumulative` constraint both over the columns and over the rows, using the `cumulative` global constraint:

```
predicate cumulative(array [int] of var int: s,
                     array [int] of var int: d,
                     array [int] of var int: r,
                     var int: b)
```

  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.
  In our formulation the constraint over the rows is written as follows:
  $cumulative(\{y_1, y_2, ..., y_n\},\ \{h_1, h_2, ..., h_n\},\ \{w_1, w_2, ..., w_n\},\ W)$
  While, the cumulative constraint over the columns is written as follows:
  $cumulative(\{x_1, x_2, ..., x_n\},\ \{w_1, w_2, ..., w_n\},\ \{h_1, h_2, ..., h_n\},\ H)$

3.0.2 Symmetry breaking constraints

There are two main symmetry cases of the solutions of the problem: the horizontal flip and the vertical flip of the plate. For this reason we decided to implement the symmetry breaking constraints through the lexicographic order between symmetrical solutions.

- **Horizontal flip**
  In the case of the horizontal flip, we implemented the lexicographical order between two arrays: $\{x_1, ..., x_n\}$ and $\{W_E - x_1 - w_1, ..., W_E - x_n - w_n\}$, where W_E is the effective width occupied of the plate, defined as the right corner of the rightest circuit in the plate.

$$lex \leq ([x_1, ..., x_n],\ [W_E - x_1 - w_1, ..., W_E - x_n - w_n]) \tag{7}$$

- **Vertical flip**
  In the case of the vertical flip, we implemented the lexicographical order between two arrays: $\{y_1, ..., y_n\}$ and $\{H - y_1 - h_1, ..., H - y_n - y_n\}$.

$$lex \leq ([y_1, ..., y_n],\ [H - y_1 - h_1, ..., H - y_n - y_n]) \tag{8}$$

The following two constraints have been inspired by the paper ”A SAT-based Method for Solving the Two-dimensional Strip Packing Problem (2SPP)” by Takehide Soh et al [1].

- **One pair of rectangles**
  We fix the positional relation between the pair of the biggest circuits, as they are the most difficult to place.
  $$lex \leq ([y_1, x_1], [y_2, x_2]) \tag{9}$$
- **Rectangles of same dimension**
  This symmetry breaking constraint considers the case in which you have two rectangles of same dimension:
  $$lex \leq ([x_1, y_1], [x_2, y_2]),\ (w_1 = w_2) \land (h_1 = h_2) \tag{10}$$

After performing some tests, the vertical flip didn’t lead to any advantage, so we decided to implement only the symmetry breaking constraint for the horizontal flip. The same result was observed for the symmetry breaking constraint for rectangles of the same size, so we decided not to implement it.

3.1 Rotation model

In order to handle the case in which there is the possibility for each circuit to be rotated, we had to manage the problem as efficiently as possible. For this reason we slightly changed the model.

3.1.1 Decision variables

In addition to the decision variables described in section 2.2, we defined other variables:

- $rotation_i$ `True` if the i^{th} circuit is rotated, `False` otherwise
- $width_i$ effective width of the i^{th} circuit on the plate
- $height_i$ effective height of the i^{th} circuit on the plate

3.1.2 Main model constraints

In addition to the *diffn* constraint described in the model without rotation, we defined other constraints. These two constraints force the rotation of a circuit if the i^{th} boolean variable of the `rotation` vector is `True`.

$$\bigwedge_{i=1}^{n} rotation_i \rightarrow ((width_i = h_i) \land (height_i = w_i)) \tag{11}$$

$$\bigwedge_{i=1}^{n} \neg rotation_i \rightarrow ((width_i = w_i) \land (height_i = h_i)) \tag{12}$$

3.1.3 Implied Constraints

We implemented the implied constraints defined in the model without rotation. However the cumulative constraint over the columns didn’t affect positively the performance, so we decided to enforce only the following constraint:

$$cumulative(\{y_1, y_2, ..., y_n\},\ \{h_1, h_2, ..., h_n\},\ \{w_1, w_2, ..., w_n\},\ W) \tag{13}$$

3.1.4 Symmetry Breaking Constraints

In addition to the symmetry breaking constraints described in the model without rotation, we defined another constraint:

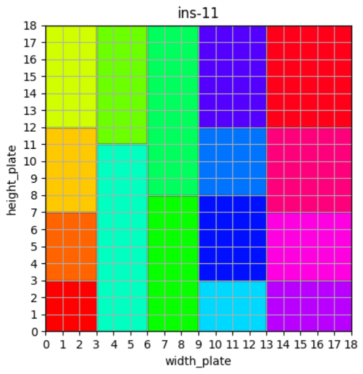
- If a circuit is a square, its rotation is useless, so we forced it not to be rotated.

$$\bigwedge_{i=1}^{n} (width_i = height_i) \rightarrow \neg rotation_i \tag{14}$$

However, in this model, the symmetry breaking constraint on the vertical flip affected positively the performance and we decided to implement it as follows:

$$lex \leq ([y_1, ..., y_n],\ [H - y_1 - h_1, ..., H - y_n - y_n]) \tag{15}$$

The figures below represent an example of the same instance solved both with circuit rotation and without.



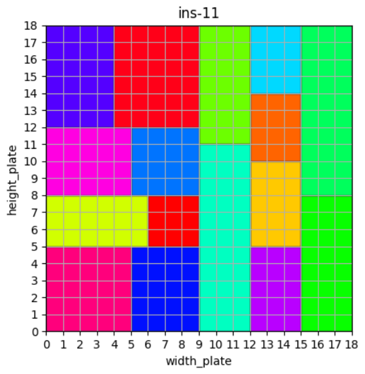


Figure 1: Instance 11 solution without (left) and with rotation (right)

3.2 Search strategy and solver

To solve this problem we made many trials over different search strategies with different solvers.
In particular we tested the sequential search both starting from the coordinates and proceeding with the height and viceversa. Moreover we tested two variable choice heuristics for the coordinates: circuits ordered by area and by height. Furthermore, we assumed that for the coordinates *indomain_min* was the best domain choice heuristic, since generally speaking it is always better to try to place a circuit in the bottom left part of the available space. In the end the best search strategy was:

```
solve
  :: int_search([H], input_order, indomain_min)
  minimize H;
```

Concerning the domain choice heuristic of H it is convenient to choose again *indomain_min*, as you want to minimize H. In this case the variable choice heuristic (*input_order*) has no direct effect because we have only one decision variable (H).
After having found the search strategy that gave the best result, we tested different solvers with the option of free search:

- Google's OR-Tools
- Chuffed
- Google's OR-Tools with free search
- Chuffed with free search
- Gecode

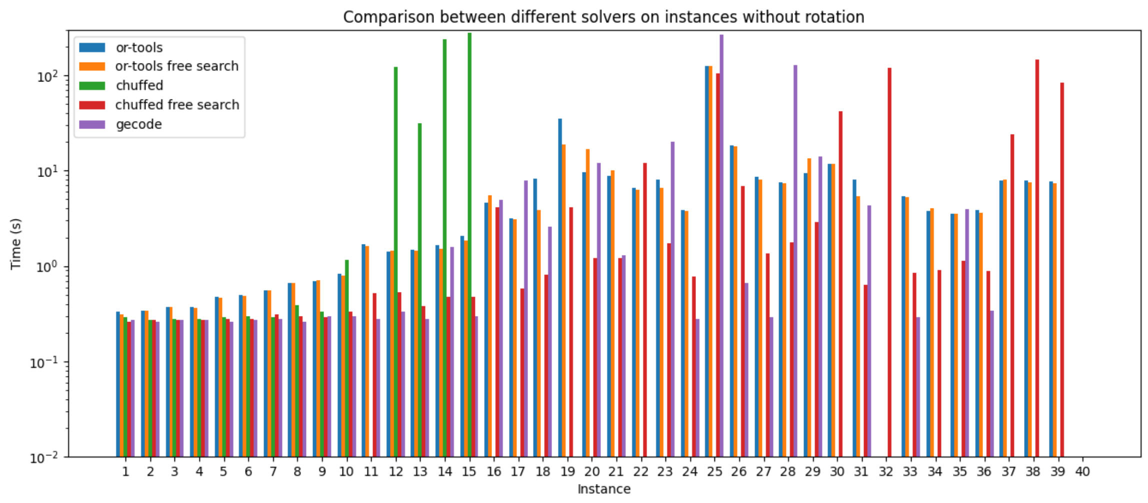


Figure 2: Performance of different solvers without rotation

In the table below are shown the number of solved instances along with the average solving time. The results refer only to the case where no rotation was allowed.

	Solved	Avg. time	Std
Chuffed	15/40	211.98	131.82
Chuffed (FS)	39/40	21.84	56.63
Or-tools	38/40	23.25	66.45
Or-tools (FS)	38/40	22.89	66.41
Gecode	33/40	79.36	127.19

Table 1: Comparing average solving time of different solvers (time in seconds, FS = free search)

From the results it is clear that Chuffed [2] with free search is the best choice, since it has solved 39 instances with an average time of 21.84 seconds.

4 Results and performances

Using the models and the search strategies we presented above we obtained the results shown in the graph below.

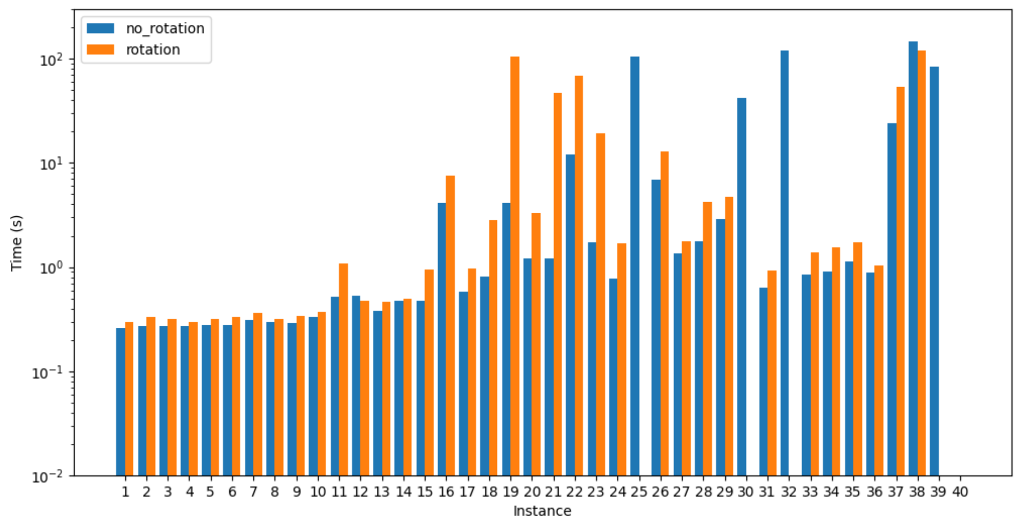


Figure 3: Performance of Chuffed with free search

As one can see from the graph, handling possible circuit rotations resulted in longer computation times and, in some cases, no optimal solution was found within the timeout. In particular, in the case where rotation wasn't allowed, it was able to solve all the instances except for 40^{th} one. On the other hand, enabling the possibility of rotating the circuits, it was able to solve all the instances except for the instances number 25, 30, 32, 39 and 40.
Below you can find in detail the solving time for each instance.

n°	No-rot	Rot
1	0.26	0.3
2	0.27	0.33
3	0.27	0.32
4	0.27	0.3
5	0.28	0.32
6	0.28	0.33
7	0.31	0.36
8	0.3	0.32
9	0.29	0.34
10	0.33	0.37
11	0.52	1.09
12	0.53	0.47
13	0.38	0.46
14	0.48	0.5
15	0.47	0.94
16	4.17	7.53
17	0.58	0.97
18	0.81	2.85
19	4.17	106.2
20	1.2	3.28
21	1.2	47.25
22	12.08	68.09
23	1.74	19.33
24	0.78	1.7
25	106.27	—
26	6.93	12.87
27	1.36	1.76
28	1.77	4.21
29	2.88	4.74
30	41.83	—
31	0.63	0.92
32	120.53	—
33	0.84	1.4
34	0.9	1.56
35	1.14	1.75
36	0.88	1.03
37	23.99	53.27
38	148.25	120.4
39	83.79	—
40	—	—

Table 2: Results (time in seconds)

4.1 Hardware

All the tests were performed on a laptop equipped with an Intel i7-1185G7 CPU, splitting the computational effort on 8 threads, as we noticed this resulted in faster computation times.

References

[1] Takehide Soh, Katsumi Inoue, Naoyuki Tamura, Mutsunori Banbara, and Hidetomo Nabeshima. A sat-based method for solving the two-dimensional strip packing problem. *Fundam. Inform.*, 102:467–487, 01 2010.

[2] Geoffrey Chu, Peter J. Stuckey, Andreas Schutt, Thorsten Ehlers, Graeme Gange, and Kathryn Francis. Chuffed, a lazy clause generation solver.