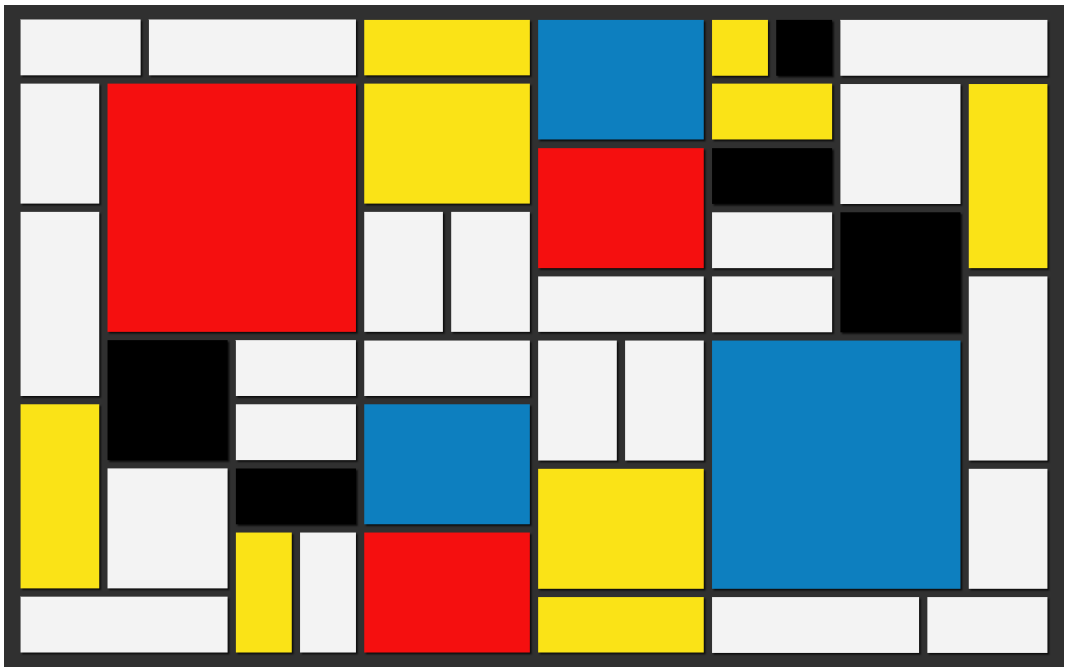
Very Large Scale Integration

Integer Linear 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

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 Integer Linear 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 number of circuits to place
- W maximum plate width
- w_i width of the i^{th} circuit
- h_i 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

With this notation a solution is represented by the set of all the couples of coordinates (x_i, y_i) for each circuit i, univocally determining its placement on the plate.

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 Basic constraints

In this section we provide a description of the basic constraints which need to be enforced in order to find a solution.

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.

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

3 ILP formulation

The ILP model that solves the VLSI problem is very simple and can be formulated through the use of the basic constraints alone. Since all the variables defined in the problem formulation are integer numbers we can directly use them to define linear constraints.

3.1 Domain Constraints

Being the domain constraints themselves linear, they can be directly encoded as they are described in (2) and (3).

3.2 No overlapping constraint

Since the no overlapping constraint contains a disjunction of four linear inequalities it cannot be directly enforced in our model, but needs to be reformulated in a purely linear form. To do this we used the Big-M notation for disjunctions as suggested in [1]. Given a large positive number M and four binary variables $d_{ijk} \quad i, j \in [1, n], i < j, k \in [1, 4]$, meaning that in (4) the k^{th} disjunction is true, the no overlapping can be written as five linear constraints:

$$\begin{aligned} x_i + w_i &\leq x_j + M(1 - d_{ij1}) \\ x_j + w_j &\leq x_i + M(1 - d_{ij2}) \\ y_i + h_i &\leq y_j + M(1 - d_{ij3}) \\ y_j + h_j &\leq y_i + M(1 - d_{ij4}) \end{aligned}$$

$$\sum_{k=1}^{4} d_{ijk} >= 1 \qquad i, j \in [1, n], i < j$$

3.3 Rotations

In order to handle the case in which there is the possibility for each circuit to be rotated, we had to define another model with some slight changes. In this model we added another set of boolean variables

$$\forall i \in [1, n] \ rot_i = \begin{cases} 1 \text{ if circuit i is rotated} \\ 0 \text{ if circuit i is not rotated} \end{cases}$$

Moreover, to take into account the fact that a circuit i is rotated we needed to swap its width and height depending of the value of rot_i. In order to achieve this result we added two new sets of variables:

- $widths_i$ which represents the actual width of the i^{th} circuit,
- $heights_i$ which represents the actual height of the i^{th} circuit.

The following additional two constraints were added to enforce the correct behaviour:

$$widths_i = w_i - rot_i * w_i + rot_i * h_i \quad \forall i \in [1, n]$$
$$heights_i = h_i - rot_i * h_i + rot_i * w_i \quad \forall i \in [1, n]$$

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

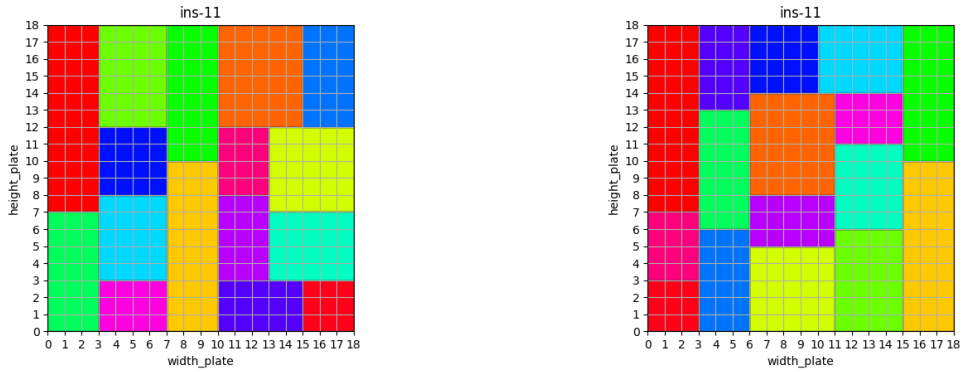


Figure 1: Instance 11 solution without (left) and with rotation (right)

3.4 Solver choice

To implement our ILP model we decided to use Google OR-Tools Python module `pywraplp`[2], which provides both simple modelling tools and a large variety of solvers, including commercial ones (such as Gurobi, CPLEX, etc.) and open source ones (such as SCIP and Google's GLOP and BOP). We performed tests with different solvers (Gurobi, SCIP, BOP) and compared the results. The first two provide a classical mixed integer linear programming solver, while BOP is a binary optimizer that uses, along with the simplex algorithm, local search and local neighborhood search. While Gurobi and SCIP have similar performances, BOP was able to solve a higher number of instances within the time limit, and so our choice fell onto it.

3.4.1 Search heuristics

Moreover, since we found the lower bound on the value of H to be very strong, we provided the solver the hint to start the search from that value, resulting in faster computation times.

4 Results and performances

Using the models and the solvers we presented above we obtained the results shown in the graphs below.

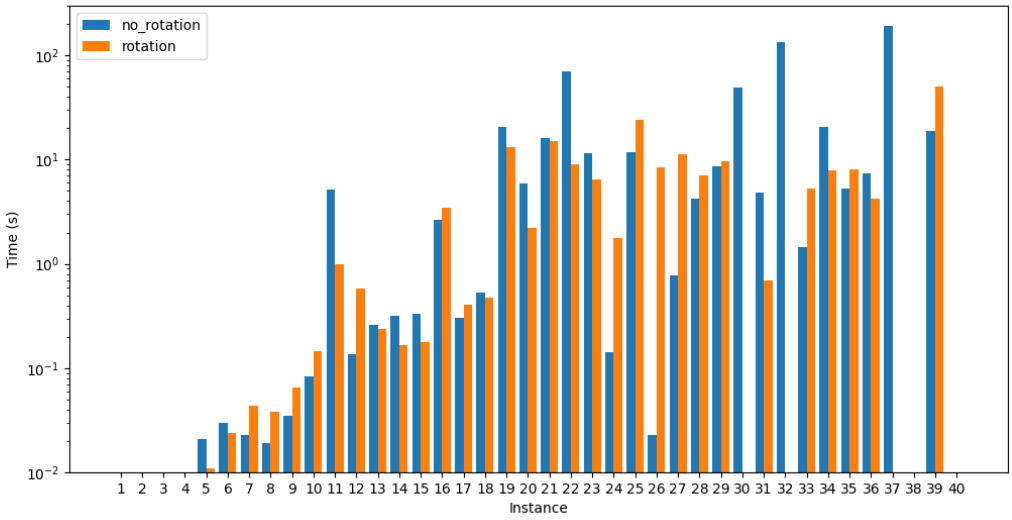


Figure 2: BOP results

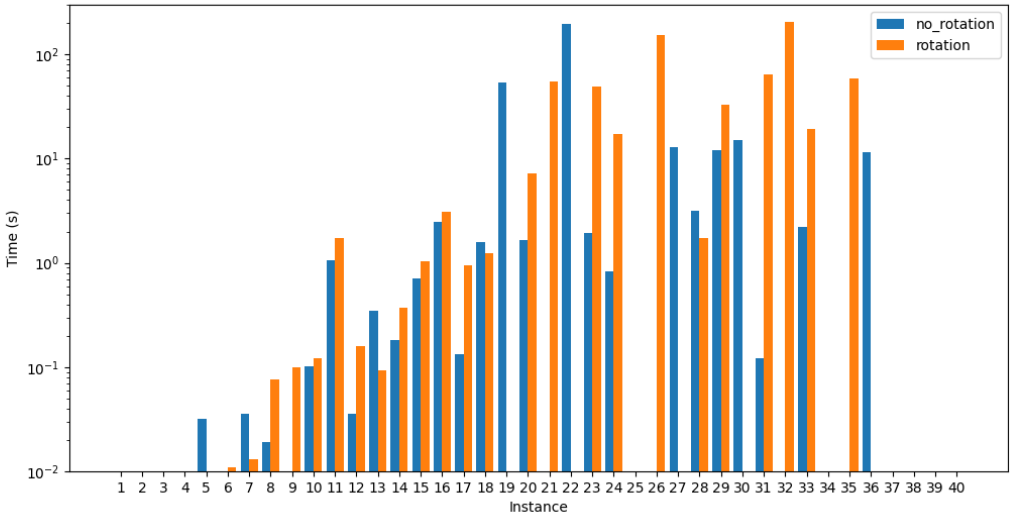


Figure 3: Gurobi results

As one can see from the graphs, handling possible circuit rotations resulted in both cases in longer computation times and, in some cases, no optimal solution was found within the timeout. As mentioned in the previous section, it is clear from the graphs that BOP outperforms Gurobi in terms of number of solved instances. In particular, in the case where rotation wasn't allowed, BOP was able to solve all the instances except for the 38^{th} and the 40^{th}. On the other hand, enabling the possibility of rotating

the circuits, it was able to solve all the instances except for the 30^{th}, the 32^{th}, the 37^{th}, the 38^{th} and the 40^{th}. On the contrary Gurobi, even in the simple case of no rotation, it was able to solve only 30 instances.

n°	No-rot	Rot	n°	No-rot	Rot
1	0.00	0.00	21	16.29	15.10
2	0.00	0.00	22	70.00	9.09
3	0.01	0.01	23	11.45	6.53
4	0.01	0.01	24	0.14	1.76
5	0.02	0.01	25	11.72	24.15
6	0.03	0.02	26	0.02	8.39
7	0.02	0.04	27	0.78	11.17
8	0.02	0.04	28	4.20	7.06
9	0.04	0.07	29	8.60	9.75
10	0.08	0.14	30	48.79	—
11	5.12	0.98	31	4.80	0.69
12	0.14	0.58	32	134.49	—
13	0.26	0.24	33	1.44	5.29
14	0.32	0.17	34	20.63	7.89
15	0.33	0.18	35	5.31	7.99
16	2.67	3.47	36	7.36	4.23
17	0.30	0.41	37	190.57	—
18	0.53	0.47	38	—	—
19	20.59	13.28	39	18.80	49.94
20	5.89	2.20	40	—	—

Table 1: Results with BOP (time in seconds)

4.1 Hardware

All the tests were performed on a desktop PC equipped with an AMD Ryzen 5 5600X CPU, splitting the computational effort on 8 threads, as we noticed this resulted in faster computation times.

References

[1] Cornell University. Big-m reformulation. https://optimization.cbe.cornell.edu/index.php?title=Disjunctive_inequalities.

[2] Google. Or-tools python reference. https://developers.google.com/optimization/reference/python/index_python.