The VLSI Design Problem

samral.tahirli@studio.unibo.it Samral Tahirli

vida.zahedi@studio.unibo.it Vida Zahedi

1. Introduction

VLSI (Very-Large-Scale Integration) design is the process of creating integrated circuits (ICs) with a large number of transistors and other components on a single silicon substrate. This project aims to address this optimization problem using various known methods. The input of the program is given in this format. Where Width indicates the width of the plate Number indicates how many pieces this input has and two lists for the width and height of each piece respectively. The goal of the model is to place a number of rectangular pieces with given widths and heights onto a plate of a given width, so as to minimize the height of the plate while ensuring that all pieces fit on it. Here we demonstrate an example for the sake of clarification.

Width = 10
Number = 6
Widths = [3,3,3,3,4,4]
Heights= [3,4,6,7,4,6]

2. CP Model

This section aims to address the VLSI Design Optimization Problem using the constraint satisfaction method and MiniZinc language, two variations of the problem are considered, in the first the pieces are fixed and can not be rotated in the second variation the pieces could be rotated by 90 degrees if decided by the solver.

2.1 Decision variables

The Decision variables of this MiniZince model and their respective domains are as follows, These domains are defined in the model to ensure that the values assigned to these variables are feasible for the problem.

- An array “x_ith[i]” which is the x-coordinate of the i-th piece on the plate. Indexes are in the range of zero to “Width” of the plate,
- An array “y_ith[i]” which is the y-coordinate of the i-th piece on the plate. Indexes are in the range of zero to MaxHeight(the sum of all Width[i] of each piece)

- OptHeight: the optimal height of the plate, Indexes are in the range of MinHeight(is the sum of the area of all the pieces divided by the width of the plate) to MaxHeight.
- An array order[i]: the order in which the pieces should be placed on the plate, based on their areas. Idexes are in the range of = 1…Number.

2.2 Objective function

The objective function of this program is to minimize the optimal height of the plate, which is defined as a variable OptHeight.

2.3 Constraints

The constraints applied in this model are as follows :

- Piece placement boundary

∀ pieces ,x_ith[i] ∈ [0, Width]

∀ pieces , y_ith[i] ∈ [0, MaxHeight]

- ensures no overlap vertically (y-axis) or horizontally (x-axis)

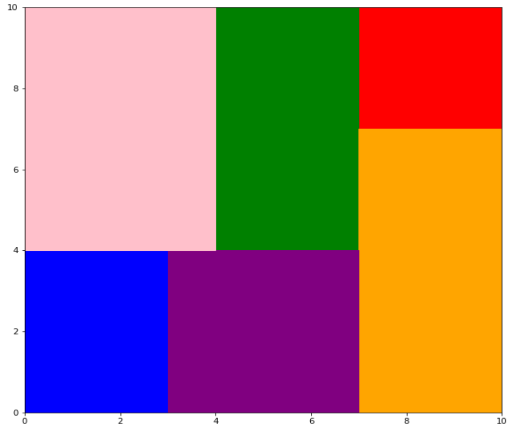
cumulative (y_ith, Heights, Widths, Width)

cumulative (x_ith,Widths, Heights, MaxHeight)

2.4 Implied constraints

- The model uses another global constraint to enforce the no overlap between the pieces this redundant constraint can speed the search procedure.

Diffn (x_ith, y_ith, Widths, Heights)

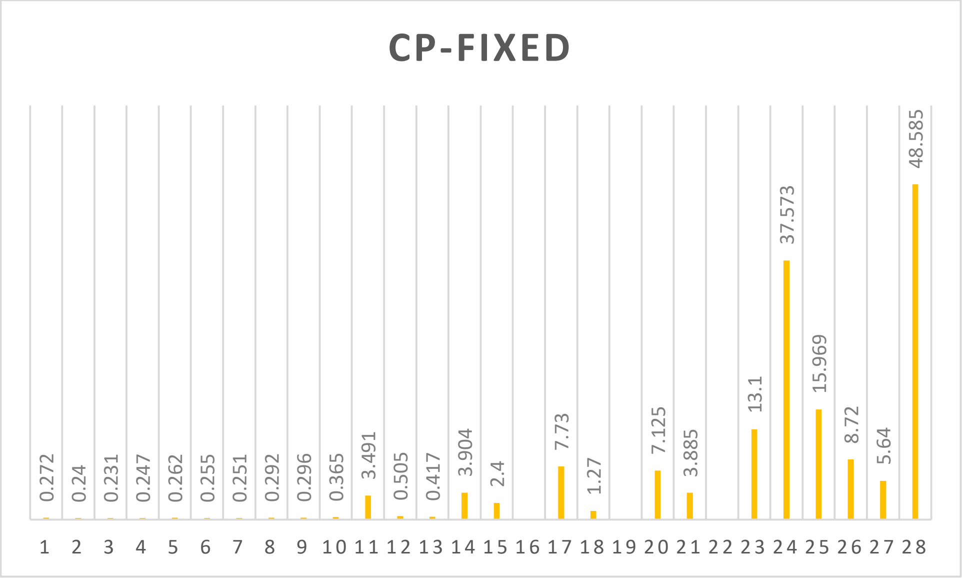


An example of the model without symmetry breaking

2.5 Symmetry breaking constraints

- In order to try to reduce the search space the pieces were ordered based on their areas and the largest piece was placed first

x_ith[order[1]] = 0 , y_ith[order[1]] = 0



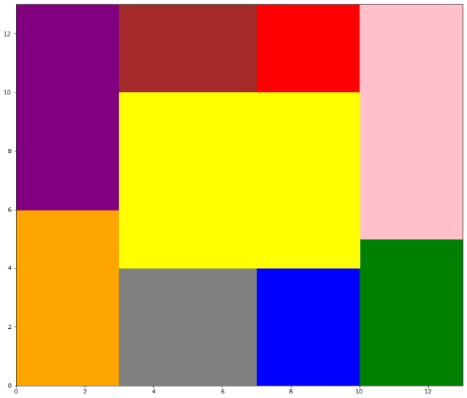
Execution time

2.6 Rotation

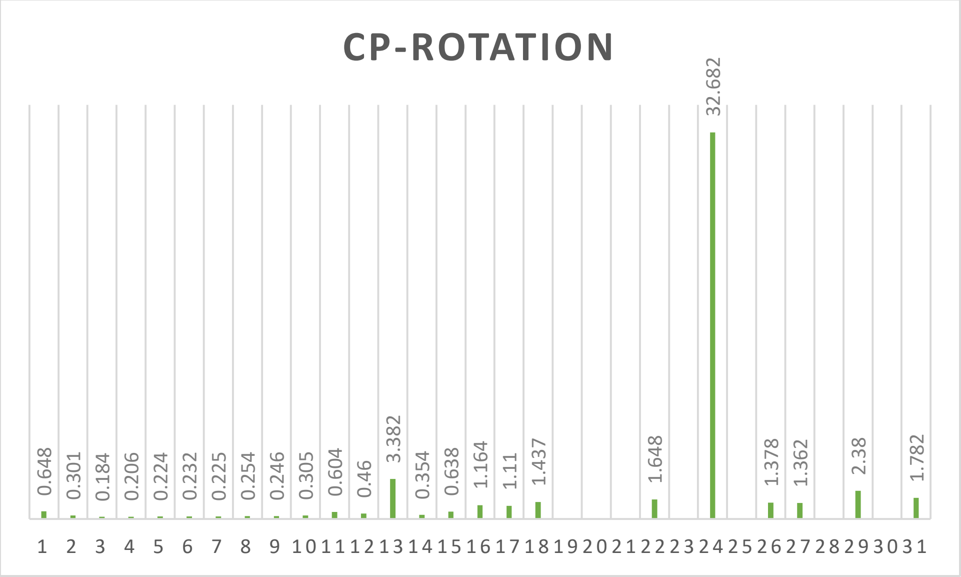
- For defining the possibility of rotation an array of Boolean decision variables was added to the model. And some other constraints were added to define the rotation as follows:

∀ pieces , widths_r[i] = Heights[i] if rotation[i] is true

∀ pieces , widths_r[i] = Widths[i] if rotation[i] is false



An example of cp model rotation version



Execution time

2.7 Validation

Experimental design

The model was run by Gecode 6.3.0 and also Chuffed 0.10.4, and a time limit of maximum 8 min was considered, the result of the two configurations are shown in the table below.

Experimental results

Chuffed and Gecode are both well-known solvers however, based on the results of our models it appears that Chuffed performs faster and also is able to solve larger input instances compared to the Gecode solver. It is noticeable that the model variant with rotation possibility using Chuffed solver can perform faster and solve the N/A instances compared to its counterpart in the first model variant.

Instaces	Chuffed-cp	chuffed-cp-r	Gecode- cp	Gecode-cp-r
6	**255 ms**	**232 ms**	**4m 55s**	**1m 31 ms**
17	7 m 3 ms	**1 m 11 ms**	N/A	N/A
22	N/A	**1 m 648**	N/A	N/A
31	N/A	**1m 782**	N/A	N/A

3.SMT Model

SMT is the problem of determining whether a mathematical formula is satisfiable. It generalizes the Boolean satisfiability problem (SAT) to more complex formulas involving real numbers, integers, and/or various data structures such as lists, arrays, bit vectors, and strings. This section aims to address this optimization problem using python language and the z3 library. For both variations mentioned earlier in the introduction.

3.1 Decision variables

The Decision variables of this SMT model and their respective domains are as follows, These domains are defined in the model to ensure that the values assigned to these variables are feasible for the problem.

- $\text{MaxHeight} = \sum (\text{Heights})$ which is used as the upper boundary.
- $\text{MinHeight} = \forall$ pieces $\sum(\text{Heights} * \text{Widths}) \,//\, (\text{Width})$.
- An array “x_ith[i]” which is the x-coordinate of the i-th piece on the plate. Indexes are in the range of zero to Number (number of pieces)
- An array “y_ith[i]” which is the y-coordinate of the i-th piece on the plate. Indexes are in the range of zero to Number (number of pieces)

3.2 Objective function

The objective function of this program is to minimize the overal height of the plate

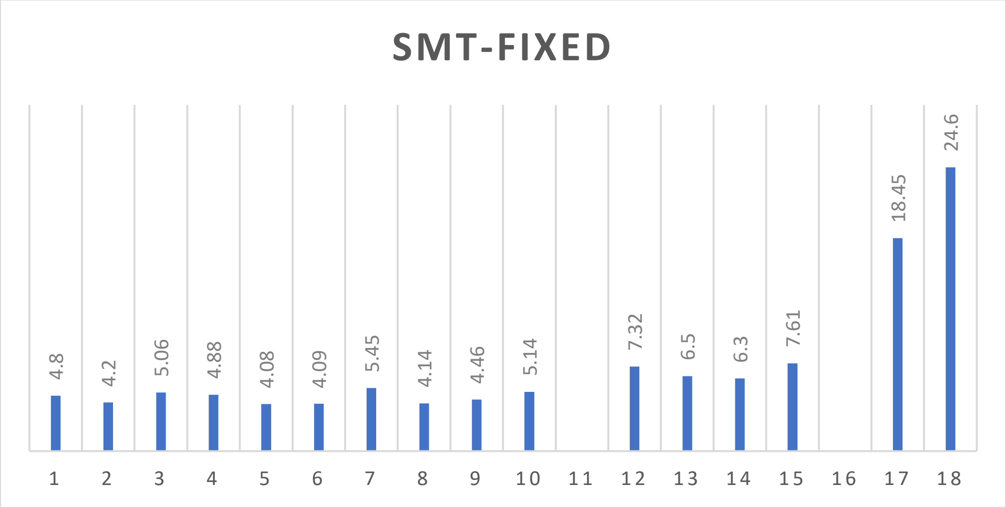
$$\forall \in \text{Pieces} : \max(\text{y\_ith}[i]+\text{Heights}[i])\ .$$

3.3 Constraints

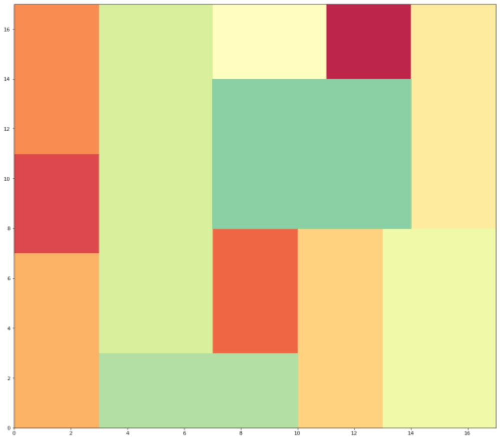
The constraints applied for the circuits to not overlap and be bounded with the width and height of plate, with the aim of minimizing the height

$$\text{diffn}(\text{x\_ith}, \text{y\_ith}, \text{Widths}, \text{Heights})$$

$$(\text{x\_ith}\ [i]+\text{Widths}\ [i] < \text{x\_ith}\ [j]) \lor (\text{x\_ith}\ [j]+\text{Widths}\ [j] < \text{x\_ith}[i]) \lor (\text{y\_ith}[i]+\text{Heights}\ [i] < \text{Heights}\ [j]) \lor (\text{y\_ith}\ [j]+\text{Heights}\ [j] < \text{y\_ith}\ [i])$$



Execution time



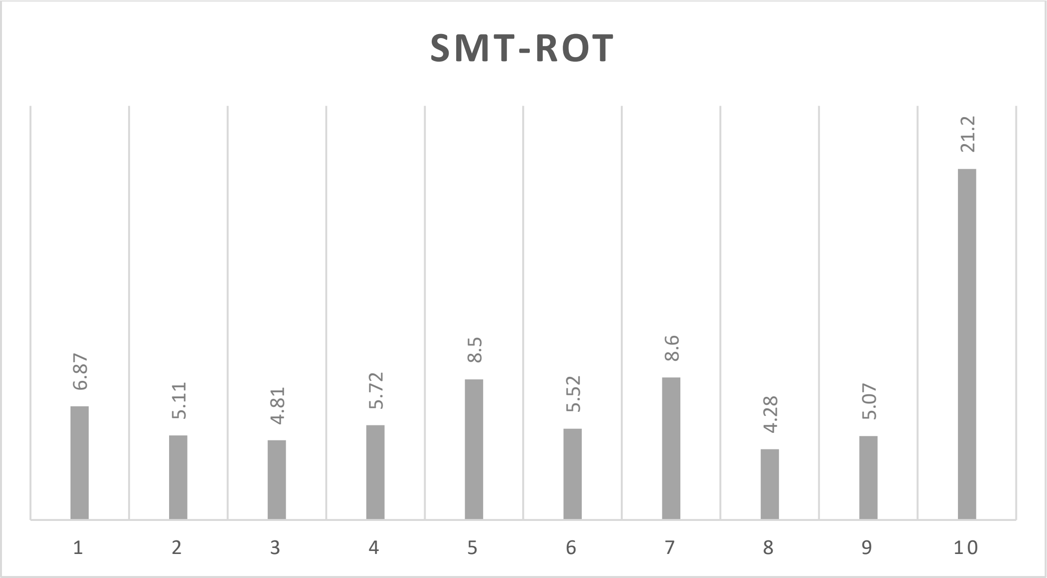
SMT FIXED IN-10

3.4 Rotation

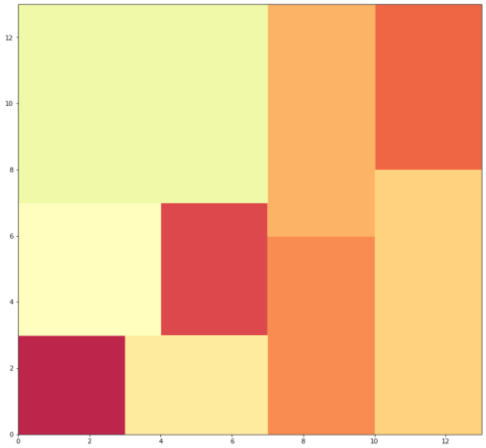
This variant of the problem introduces 3 more lists one for rotation possibility and two for storing the Widths and Heights of the pieces after rotation, calling them Widths_r and Heights_r respectively. (If a piece is rotated, then its width becomes its height and its height), added constraints are as follows

$$\forall \text{ Pieces : Rotation}[i] \iff (\text{Widths\_r } [i] = \text{Heights}[i]) \land (\text{Heights\_r } [i] = \text{Widths}[i])$$

$$\text{diffn}(\text{x\_ith}, \;\text{y\_ith}, \text{Widths\_r}, \text{Heights\_r})$$



Execution time



SMT-ROTATION INS-6

3.5 Validation

Experimental design

The model is defined in google collab in form of a jupyter notebook,the execution time for each instance is being calculated using time library, a time limit of 5 minute is being considered.

Experimental results

The fixed model seems to perform better for some of the instances, while the rotation variant tends to reduce the execution time for solving same instances. The table is provided for comparison bellow.

Instance	SMT-FIXED	SMT-ROT
4	7.77s	5.37
10	7.62s	12.49
18	25.17	19.14
20	N/A	N/A
24	30.96	59.02

3. MIP Model

Mixed-integer linear programming (MILP) is often used for system analysis and optimization as it presents a flexible and powerful method for solving large, complex problems such as the case of industrial symbiosis and process integration. This format of programming only allows constraints in form of mathematical equations and inequalities. We consider to variants of the VLSI problem one with fixed width and one with the possibility of rotation.

4.1Decision variables

The decision varaiables are as follows

- OptHeight which is the optimal height of the plate
- Two lists x and y
- A comb value combination of boolean type

4.2 Objective function

The objective of this mode is to minimize the height of the plate.

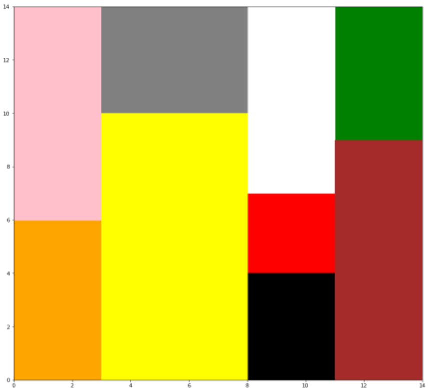
4.3 Constraints

The constraints for MILP programming should be in form of linear equations and inequalities, and also there is no is logical OR in linear programming. To solve this issue, there is a technique called "big-M trick", which we tried to implement in the code

∀∈ Pieces x + x_ith <= Width
∀∈ Pieces y + y_ith <= OptHeight

The big M trick is applied as follows : M1 and M2 should have big value like 10**6

x[j] + x_ith[j] <= x[i] + M1 * (1 - comb[i,j,0])
x[i] + x_ith[i] <= x[j] + M1 * (1 - comb[i,j,1])
y[j] + y_ith[j] <= y[i] + M2 * (1 - comb[i,j,2])
y[i] + y_ith[i] <= y[j] + M2 * (1 - comb[i,j,3])



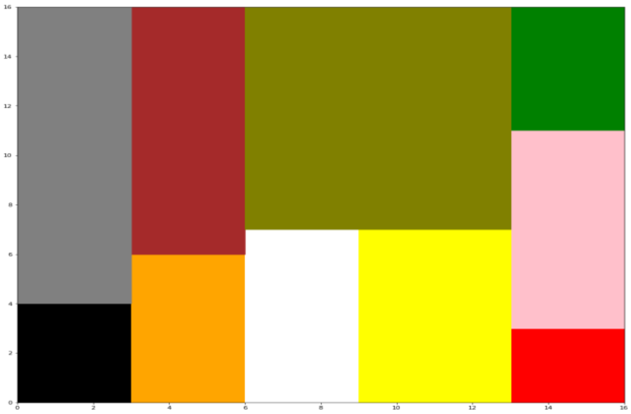
MILP-FIXED MODEL INS7

4.4 Rotation

This variation of the model also takes into account the possibility of rotating the pieces by introducing a numpy array for rotation and two lists for storing the rotated widths and heights respectively.

These constraints will be added to the model

```
x[j]  + Rotation[j, 0] <= x[i]  + M1 * (1 - comb[i,j,0])
x[i]  + Rotation[i, 0] <= x[j]  + M1 * (1 - comb[i,j,1])
y[j] + Rotation[j, 1] <= y[i]  + M2 * (1 - comb[i,j,2])
y[i] + Rotation[i, 1] <= y[j]  + M2 * (1 - comb[i,j,3])
```



MILP-ROT MODEL INS9

4.5 Validation

Experimental design

The model uses python language and gurobipy for defining the model. The model was implemented in google collab in form of a jupyter notebook, for the sake of simplicity a time limit of five minutes was considered for the execution time of each instance, if the instance exceeds this time, we stop the execution, and place N/A for the output value

Experimental results

As expected the rotation possibility will reduce the execution time for solving the instances, however, there are some exceptions, for example, the rotation model was not able to solve instances 24 and 26 within the duration of 5 minute. bellow there is a small comparison between the two models.

Instance	MILP-FIXED	MILP-ROT
5	0.104s	0.21
13	17.8s	4.49
24	3.52s	N/A
26	N/A	N/A

4. Conclusions

It is evident that the best interface for defining and solving the optimization and constraint programming would be minizinc, since this language has some already defined global constraints in which are quite handy for solving this kind of problem, on the other hand Other interfaces like z3-solvers and gurobipy do not contain these constraints beforehand therefore we have to define them from scratch which could become quite complex and time consuming at times, comparing the models we define here the cp and its variant are able to solve more instances compared to others mentioned above, thus the best approach would be to use minizinc and constraint programming for solving such problems