VLSI Design Project
Combinatorial Decision Making and Optimization 2021/2022

Memoona Shah(0001056334),
Daniele Baiocco(0000000000),
Pratiksha Pratiksha(0000000000)

Contents

1 Introduction **4**

2 Constraint Programming **4**
2.1 CSP: Overview . . . 4
2.2 MINIZINC . . . 4
2.3 Decision Variables . . . 4
2.4 Constraints . . . 4
2.4.1 Constraint 1: all different . . . 4
2.4.2 Constraint 2: Geoff . . . 4
2.4.3 Constraint 3: Cumulative Constraints . . . 4
2.4.4 Constraint 4: . . . 4
2.4.5 Constraint 5: . . . 4
2.4.6 Constraint 6: . . . 4
2.5 Objective Function . . . 4
2.6 Symmetry Breaking . . . 4
2.7 Rotation . . . 4
2.8 Results . . . 4
2.9 Conclusion . . . 4
2.10 References . . . 4

3 SAT **5**
3.1 SAT: Overview . . . 5
3.2 Decision Variables . . . 5
3.3 Constraints . . . 5
3.3.1 Constraint 1: all different . . . 5
3.3.2 Constraint 2: Geoff . . . 5
3.3.3 Constraint 3: Cumulative Constraints . . . 5
3.3.4 Constraint 4: . . . 5
3.3.5 Constraint 5: . . . 5
3.3.6 Constraint 6: . . . 5
3.4 Objective Function . . . 5
3.5 Symmetry Breaking . . . 5
3.6 Rotation . . . 5
3.7 Results . . . 5
3.8 Conclusion . . . 5
3.9 References . . . 5

4 SMT **6**
4.1 SMT: Overview . . . 6
4.2 Z3 and Python . . . 6
4.3 Modelling the problem . . . 6
4.3.1 Maximum height of the circuit board . . . 6
4.4 Decision Variables . . . 6
4.5 Constraints . . . 7
4.5.1 Constraint 1: All Different Constraint . . . 7
4.5.2 Constraint 2: Non-Overlapping Constraint . . . 7
4.5.3 Constraint 3: Cumulative Constraints . . . 7
4.5.4 Constraint 4: Domains of px_i and py_i . . . 7

4.5.5 Constraint 5: Symmetry Breaking Constraint . . . 8
4.6 Objective Function . . . 8
4.7 Rotation . . . 9
4.8 Results . . . 12
4.9 Conclusion . . . 13

5 Linear Programming 15
5.1 CSP: Overview . . . 15
5.2 MINIZINC . . . 15
5.3 Decision Variables . . . 15
5.4 Constraints . . . 15
5.4.1 Constraint 1: all different . . . 15
5.4.2 Constraint 2: Geoff . . . 15
5.4.3 Constraint 3: Cumulative Constraints . . . 15
5.4.4 Constraint 4: . . . 15
5.4.5 Constraint 5: . . . 15
5.4.6 Constraint 6: . . . 15
5.5 Objective Function . . . 15
5.6 Symmetry Breaking . . . 15
5.7 Rotation . . . 15
5.8 Results . . . 15
5.9 Conclusion . . . 15

6 Comparative analysis of the results in CSP, SAT/SMT and LP 16

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). This enabled the modern cellphone to mature into a powerful tool that shrank from the size of a large brick-sized unit to a device small enough to comfortably carry in a pocket or purse, with a video camera, touchscreen, and other advanced features.

2 Constraint Programming

2.1 CSP: Overview

2.2 MINIZINC

2.3 Decision Variables

2.4 Constraints

2.4.1 Constraint 1: all different

2.4.2 Constraint 2: Geoff

2.4.3 Constraint 3: Cumulative Constraints

2.4.4 Constraint 4:

2.4.5 Constraint 5:

2.4.6 Constraint 6:

2.5 Objective Function

2.6 Symmetry Breaking

2.7 Rotation

2.8 Results

2.9 Conclusion

2.10 References

3 SAT

3.1 SAT: Overview

3.2 Decision Variables

3.3 Constraints

3.3.1 Constraint 1: all different

3.3.2 Constraint 2: Geoff

3.3.3 Constraint 3: Cumulative Constraints

3.3.4 Constraint 4:

3.3.5 Constraint 5:

3.3.6 Constraint 6:

3.4 Objective Function

3.5 Symmetry Breaking

3.6 Rotation

3.7 Results

3.8 Conclusion

3.9 References

4 SMT

4.1 SMT: Overview

In computer science and mathematical logic, satisfiability modulo theories (SMT) is the problem of determining whether a mathematical formula is satisfiable. An SMT instance is a generalization of a Boolean SAT instance in which various sets of variables are replaced by predicates from a variety of underlying theories. SMT formulas provide a much richer modeling language than is possible with Boolean SAT formulas. The difference is that SMT solvers takes systems in arbitrary format, while SAT solvers are limited to Boolean equations in CNF1 form

4.2 Z3 and Python

In the solution of VLSI Circuit arrangement solution we have used Z3 library in Python. Z3 is a theorem prover from Microsoft Research with support for bitvectors, booleans, arrays, floating point numbers, strings, and other data types. Z3 is used for both SAT as we did in out previous solution and SMT based solution approaches. In the first one we only use boolean atoms and their CNF form of clauses to solve any problem. Where as in SMT, it is possile to solve a problem using domain based structures and predicates.

4.3 Modelling the problem

4.3.1 Maximum height of the circuit board

The worst possible scenario can be where all the circuits are one top of another and therefore the maximum height can be the sum of all the circuits heights. Lets say:

$height_{max} = \sum_i^n h_i$ where $i = 1, 2, 3...n$ and n is total number of circuits.

And h_i is the height of of each circuit to be placed.

4.4 Decision Variables

To solve this problwm with SMT, first we need to define the variables we need to model our porοblem. It is possible to create variables of different kinds such as boolean, constants, real numbers, integers, BITwise vectors, floating point values and arrays. The variables used in this solution are mainly integeres and are stated as the following:

1. **X-coordinate position:** p_x: An integer value to be assigned to each px_i variable that will be circuit i's horizontal position in the circuit board.

$$\forall i \ \epsilon \ \{1...n\} : Px_i \ \epsilon \ \mathbb{Z}$$

2. **Y-coordinate position:** p_y: An integer value to be assigned to each py_i variable that will be circuit i's vertical position in the circuit board.

$$\forall i \ \epsilon \ \{1...n\} : Py_i \ \epsilon \ \mathbb{Z}$$

(p_x and p_y are the bottom left corner position of each circuit in the VLSI Board.)

3. **Maximum height - H**

$$H = 2 * max[max(h_i), \frac{\sum_i h_i w_i}{W}]$$

4.5 Constraints

4.5.1 Constraint 1: All Different Constraint

Basically, we dont want our (px,py) position for two circuits to be the same. This partly solves our non-overlap problem. The builtin predicate of Z3 *Distinct* is used to achieve this constraint implementation. The constraint is:

For all n circuits we apply:

$$Distinct(z_1, z_2, z_3...z_n)$$

Where z_i is a vector combinition of px_i and py_i.

4.5.2 Constraint 2: Non-Overlapping Constraint

For each pair of circuits n_i and n_j where $1 <= i < j < n$, the non overlapping constraint is:

$$(px_i + w_i \leqslant px_j) \lor (px_j + w_j \leqslant px_i) \lor (py_i + h_i \leqslant py_j) \lor (py_j + h_j \leqslant py_i)$$

4.5.3 Constraint 3: Cumulative Constraints

In minizinc, we used a global cumulative constraint to solve this problem. Similar approach has been implemented in this part also. The cumulative constraint is implemented twice once for x-axis and once for y axis. For x-axis, by considering the start, duration, recources and total recources to be horizontal position , widths, heights and plate height respectively. For y-axis, by considering the start, duration, recources and total recources to be vertical position , heights, widths and plate width respectively.

The cumulative Scheduling constraint is given by *cumulative(S,d,r,c)*. This constraint basically enforces that at each point in time t, the jobs running at this time t must not exceed its capacity. Mathematically:

$$\sum_{S_i \leq r \land r < S_i + d_i} r \ \leq \ c \qquad forall \ \ i \ \epsilon \ \{1, 2..n\}$$

4.5.4 Constraint 4: Domains of px_i and py_i

The domains of px_i and py_i are declared as follows:

$$\begin{aligned} D(px_i) \ &= \ \{ \ c \ \epsilon \ \mathbb{N} \mid 0 \ \leq c \ \leq W - w_i \} \\ D(py_i) \ &= \ \{ \ c \ \epsilon \ \mathbb{N} \mid 0 \ \leq c \ \leq H - h_i \} \end{aligned}$$

4.5.5 Constraint 5: Symmetry Breaking Constraint

The symmetries possible in this problem can be of two kinds as we have already discussed in previous sections also. They are horizontal flip and 180 degrees rotation. In order to break these symmetries we applied the following constraints.

1. **Horizontal flip**: A constraint was implemented that enforces that for all those circuits whose horizontal coordinates lie on the left half part of the circuit board, must have the sum of area covered by it to be greater or equal to the ones on the right half side of the board. Mathematically:

   For all $i \epsilon \{1..n\}$

$$\sum_{i \wedge \ (px_i \leq \frac{W}{2})}^{n} A_i \geq \sum_{i \wedge \ (px_i > \frac{W}{2})}^{n} A_i$$

where A_i is the area of the circuit i calculated by (height x width).

$$\vee_i$$

2. The longest circuit (i.e with biggest height) among all is put at (0,0) by default. Mathematically;

$$(px_m, py_m) = (0, 0)$$

$$\begin{array}{rl} \text{where} & px_m = max(px_1, ..., px_n) \\ \text{and} & py_m = max(py_1, ..., py_n) \end{array}$$

3. Another constraint that we implemented in this is Left-Right and UP down constraint. This constraint is the same as discussed and implemented in the SAT section also. It basically is as following: For each pair of circuits n_i and n_j where $1 <= i < j < n$, the constraint is:

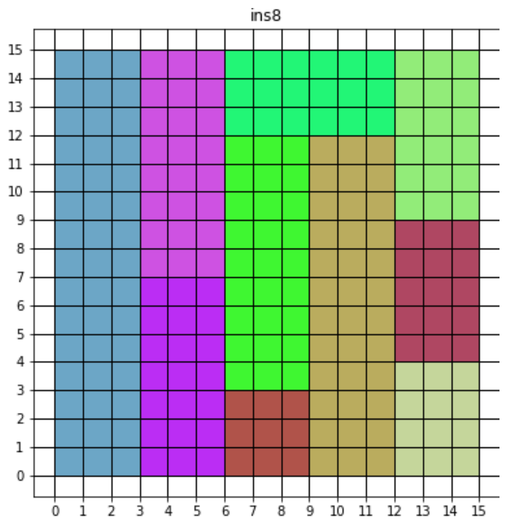
$$(Zlr_{ij} \wedge px_i + w_i \leqslant px_j) \vee (Zlr_{ji} \wedge px_j + w_j \leqslant px_i) \vee (Zud_{ij} \wedge py_i + h_i \leqslant py_j) \vee (Zud_{ji} \wedge py_j + h_j \leqslant py_i)$$

   where Zlr_{ij} is true if the circuit i is to the left of circuit j. And similarly, Zud_{ij} is true if circuit i is above circuit j, and vice versa.

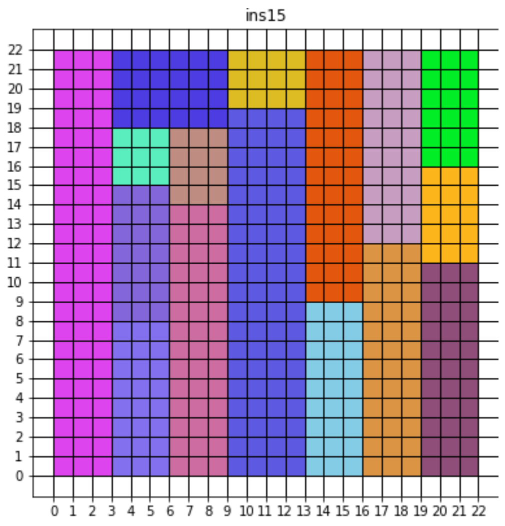
4.6 Objective Function

The goal of this project was (a) to find optimal assignments to all circuit positions and (b) to minimize the maximum height of the VLSI board. Therefore, we used *model.optimize()* Z3 function to define our objective function. This instructs the solver to minimize the variable assigned to it.

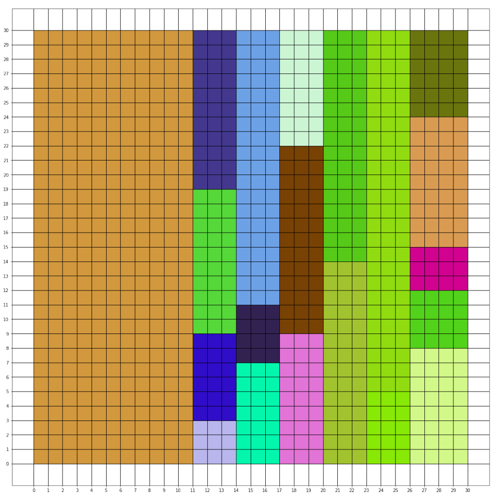
```
opt = Optimize()
opt.minimize(length)
```



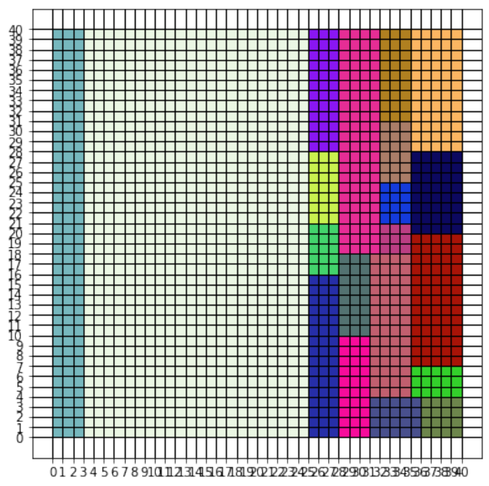
(a)



(b)



(c)



(d)

Figure 1: Results of SMT

4.7 Rotation

In this part of the problem we allow the rotation of circuits which means the optimizer can rotate a circuit by a degree of 90 by swapping their respective height and width. For this a new array with the name rotation is created to apply the new possibilities. All the constraints remain the same as the previous model without the rotations with this addition. The new array created relate to the real dimensions after a possible swap in the following way.

$$\text{forall i } \epsilon \text{ \{ 1..n\}:}$$
$$x\_r_i = if(rotation_i == True, y_i, x_i)$$
$$y\_r_i = if(rotation_i == True, x_i, y_i)$$

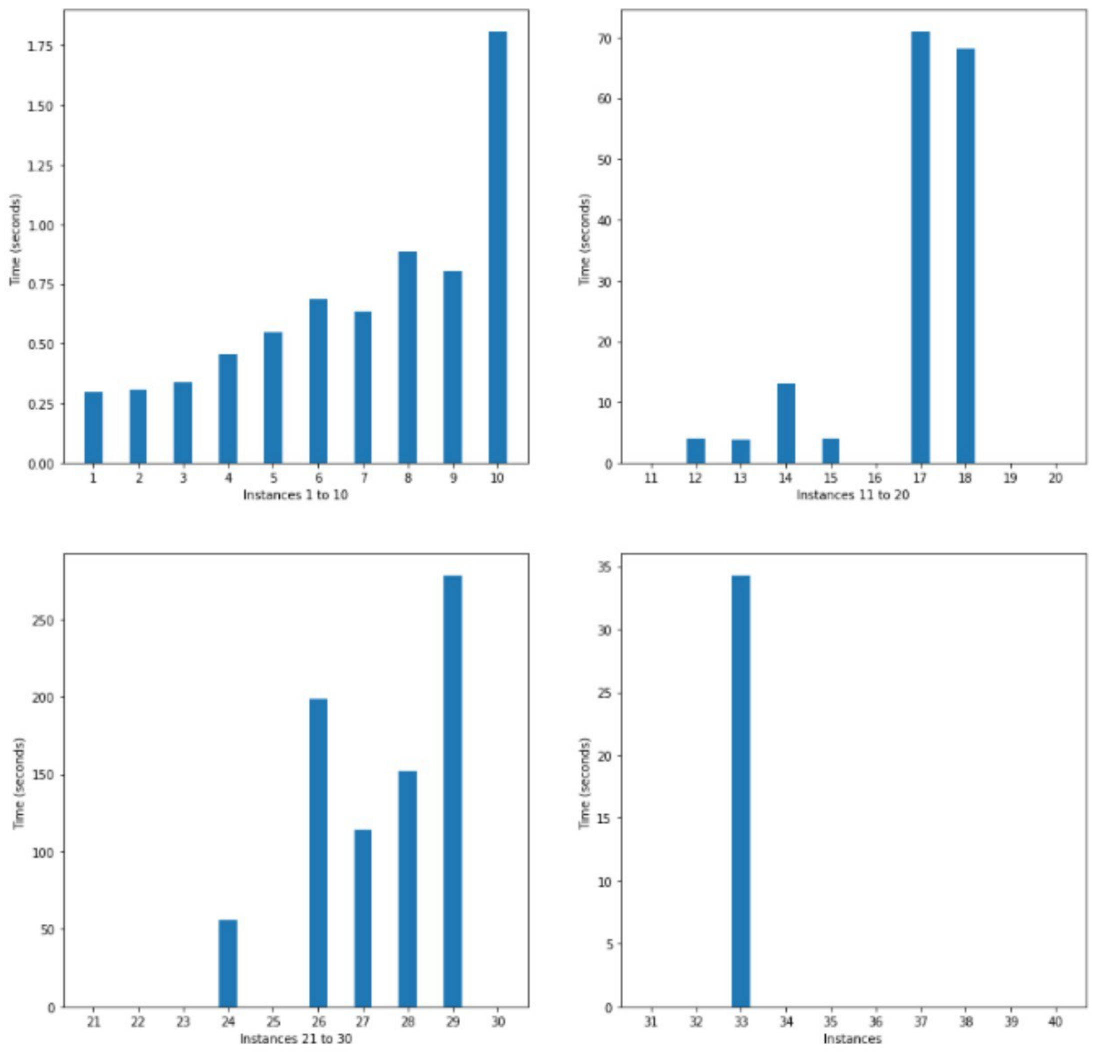


Figure 2: Results of SMT without rotation

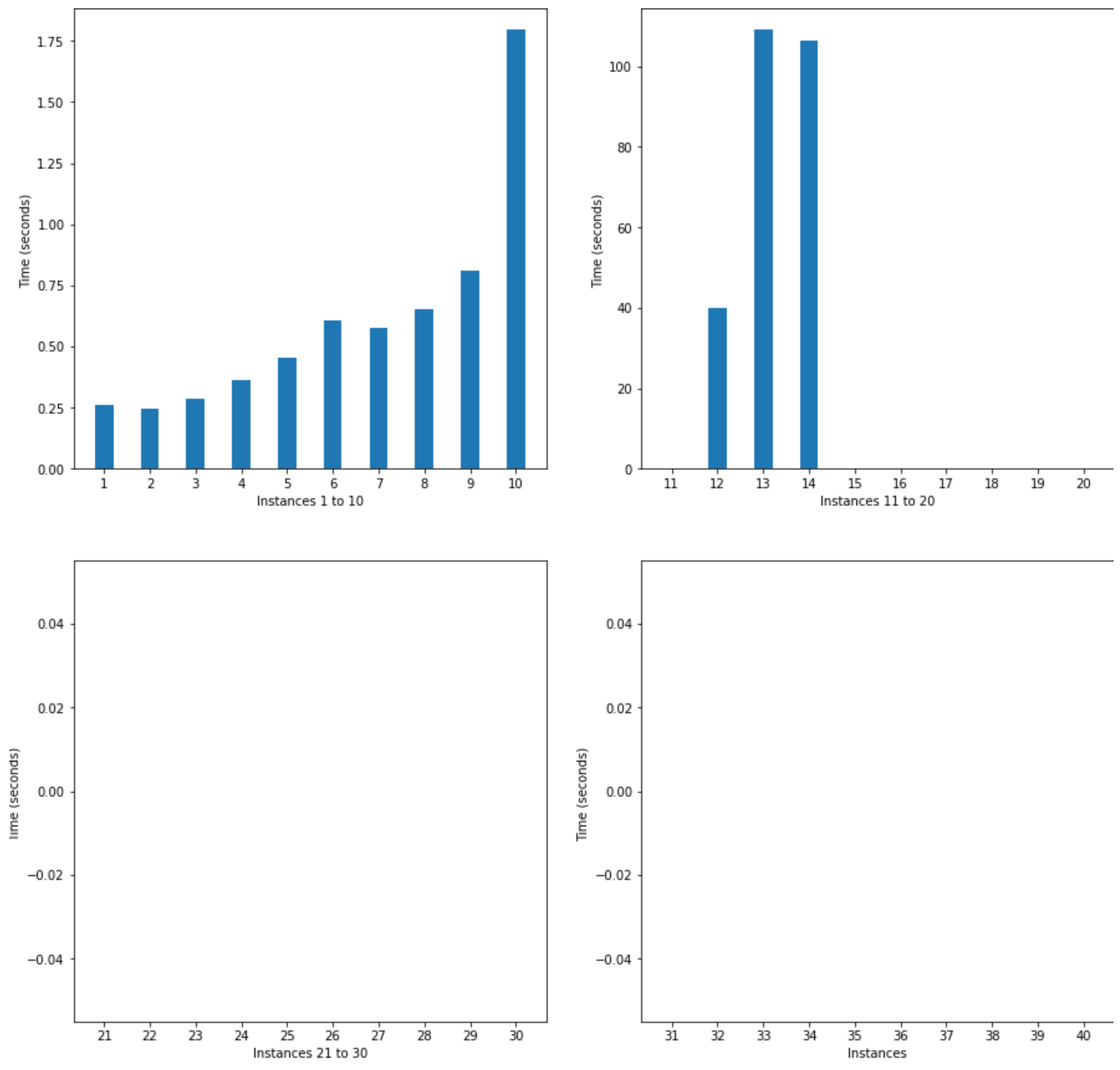


Figure 3: Results of SMT with rotation

4.8 Results

4.9 Conclusion

In this solution, we started from creating the decision variables and then we implemented some constraints on those variables to solve the problem. In the process we also defined the domains for some variables to reduce the search space. Among the forty instances used to check the model, we have solved twenty two instances. As about the speed, as can be seen in figure 2, for the first ten instances speed is good. However, for the other thirty instances, some easier instances were not solved in the time constraint of 300 seconds where as a difficult instance such as instance 33 as shown in figure 1 is shown was solved in short time.

References

[1] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB), www.SMT-LIB.org. 2016

[2] A. Steinberg, A strip-packing algorithm with absolute performance bound 2 SIAM J. Comput., 26 (1997), pp. 401-409, 10.1137/S0097539793255801

[3] Cid-Garcia NM, Rios-Solis YA (2021) Exact solutions for the 2d-strip packing problem using the positions-and-covering methodology. PLoS ONE 16(1): e0245267. https://doi.org/10.1371/journal.pone.0245267

[4] Rectangle packing in practice. Stoykov, P. B. (Author). 25 Sep 2017 Student thesis: Master

[5] Yurichev's book "SMT by Example" https://yurichev.com/writings/SAT_SMT_by_example.pdf

[6] Hakank's examples http://www.hakank.org/z3/

5 Linear Programming

5.1 CSP: Overview

5.2 MINIZINC

5.3 Decision Variables

5.4 Constraints

5.4.1 Constraint 1: all different

5.4.2 Constraint 2: Geoff

5.4.3 Constraint 3: Cumulative Constraints

5.4.4 Constraint 4:

5.4.5 Constraint 5:

5.4.6 Constraint 6:

5.5 Objective Function

5.6 Symmetry Breaking

5.7 Rotation

5.8 Results

5.9 Conclusion

6 Comparative analysis of the results in CSP, SAT/SMT and LP

From our experiments we can conclude that . . .