Natural Language Processing
VLSI Design

Rasha Jabbour - rasha.jabbour@studio.unibo.it

February 2023

1 Introduction

This report describes three different approaches to the VLSI problem. These approaches are Constraint Programming (CP), Satisfiability Modulo Theories (SMT), and Mixed-Integer Linear Programming (MIP).

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.

The purpose is to design the VLSI of the circuits defining their electrical device: 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 × m circuit cannot be positioned as an m × n circuit in the silicon plate. In the second case, the rotation is allowed, which means that an n × m circuit can be positioned either as it is or as m × n.

For this project I decided to do it on my own although I know that was not recommended. I took this decision because along with my studies I am also working and I preferred to not have to depend on others or have others depend on me as my timing could be erratic and I did not want to have to change exam dates while in a team. It took me around 2 months to be able to finish the whole project on my own.

This of course came with its own set of issues for me as it was a very big project for one person and so I have a few approaches that work better than others as it was hard for me to give all three approaches the same level of attention. Rotation also gave me a few issues for all of the approaches.

My laptop specs are:
RAM: 8 GB DDR4
CPU: Intel core i5 9th gen
GPU: 940MX
Windows 11
I used Thonny as my IDE.

1.1 Approach

For all these approaches I used a base model for rotate and one for non-rotate and I used the same utils.py script for all three as well.

Using the example given in the project given:
w = 9
n = 5
cx = [3,2,2,3,4]
cy = [3,4,8,9,12]
where:
• w represents the width of the silicon plate;
• n is the number of necessary circuits to place inside the plate;
• cx and cy are two arrays, indexed from 1 to n, containing the horizontal and vertical sizes of each circuit (for example the first circuit is of size 3x3).

I also set the Decision Variables X and Y which are the Horizontal and Vertical coordinate of the circuits:
X is the range from 0 till the width of the plate
Y is the range from 0 till the upper bound that will be introduced below

The lower and upper bounds given here for the normal models are:

$$lower = \max\left(\max\left(cy\right), \left\lceil \frac{\sum_{i\in circ}\left(cx_i \cdot cy_i\right)}{w} \right\rceil\right) \tag{1}$$

$$upper = \sum_{i=1}^{n} c_{y_i} \tag{2}$$

2 CP Model

2.1 Decision variables

The decision variables are w, n, cx, cy, x, y, lower, upper, and h.
h ranges from the below bounds:

$$lower \ldots upper : h \tag{3}$$

Refer to Section 1.1 for the rest.

2.2 Objective function

Our purpose is to find the minimum height needed for the plate to fit all the circuits that also stays within the bounds provided. I used the bounds given in Section 2.1 for the height.

2.3 Constraints

The model contains constraints to ensure that all circuits fit on the plate and do not overlap. It also includes cumulative constraints to ensure that the width and height of each circuit do not exceed the width and height of the plate, respectively. Additionally, the diffn constraint is used to ensure that the circuits do not overlap.

The code also includes symmetry breaking constraints to ensure that the biggest circuit is always placed in the bottom left part of the plate and the second biggest circuit is placed under and on the left of the biggest circuit.

The code also includes a time limit for the solver, to prevent it from running longer than 5 minutes.

2.3.1 Cumulative Constraints

Cumulative is an inbuilt Minizinc Constraint. The first parameter passed to cumulative is typically an array with the start times of each task, the second is an array with the durations of each task, the third is an array with the resource requirements for each task, and the fourth is an array with an upper bound on the resources available.

$$constraint \quad cumulative(y, cy, cx, w) \tag{4}$$

$$constraint \quad cumulative(x, cx, cy, h) \tag{5}$$

These two constraints specify that the circuit is represented as an activity with its duration equal to its height and the amount of resource it requires equal to its width. The first constraint,equation 4, states that the cumulative height of all activities along the Y-axis (represented by ”y”) should not exceed the capacity of the resource along that axis (represented by ”cy”). The second constraint, equation 5, states that the cumulative width of all activities along the X-axis (represented by ”x”) should not exceed the capacity of the resource along that axis (represented by ”cx”).

2.3.2 Domain Reduction Constraints

This code specifies two constraints for the variables x at instance i and y at instance that loop over the range of n. The first constraint states that each x at i must be less than or equal to the result of the subtraction of the plate width with the value of the circuit width at i, and the second constraint states that each y at i must be less than or equal to the result of the subtraction of the plate height with the value of the circuit height at i. The constraints are declared as part of a domain in the code. This indicates that these constraints define a domain or a set of feasible values for the variables x at i and y at i.

2.3.3 No Overlap Constraint

The key concept was to establish a relationship between the beginning and end of each pair of circuits for both coordinates in order to prevent potential circuit overlaps. The decision was made to apply the global constraint diffn, which mandates the non-overlapping of rectangles given their origin points and sizes, after briefly considering the packing constraints described in the Minizinc library. This is the ideal solution to our circuit positioning issue.

2.3.4 Symmetry Breaking Constraint

The solver has access to numerous symmetric variations of a given solution. Applying symmetry breaking constraints is a smart strategy to decrease the number of solutions. In its most basic form, a symmetry breaking constraint consists of adding restrictions to the model that exclude all symmetrical iterations of a (partial) assignment to the variables except for one. These restrictions are known as static symmetry breaking restrictions. The fundamental purpose of symmetry breaking is to impose order. In our circumstance, we made the decision to always place the largest circuit in the plate's lower left corner. Additionally, the second-largest item must always be positioned to the right of or atop the largest item.

I defined an array ordered circuits by sorting the elements in circ based on a custom comparison function.

$$-cy[c] \cdot cx[c] \quad such that \quad c \in circ \tag{6}$$

The comparison function is defined as equation 6, which computes the product of cx at c and cy at c for each element c in circ and negates the result. The sort by function sorts circ in ascending order based on the comparison function.

In other words, this expression sorts circ based on the product of cx at c and cy at c in descending order, and assigns the sorted array to ordered circuits.

The constraint can then be set after taking the size of the circuits into account. On the coordinates of the circuits, we use the global constraint lex lesseq to achieve that. The array x must be lexicographically less than or equal to the array y in order to satisfy this condition. Regardless of indices, it compares them from first to last element. Since we already used diffn to ensure that each instance has unique coordinates, we can use the lexicographically order. One of the two coordinates, however, might be shared by two circuits. Because of this, the less equal operator is required rather than the less not equal. This would have been a difficult constraint in the latter scenario.

2.4 Rotation

The lower and upper bounds for the CP rotation model are the same as the ones for the normal model

I also introduced a Boolean variable called isrot that whether a circuit is rotated or not. isrot is True if it is rotated and False if not.

Furthermore, added two variables cxr and cyr that represents the actual horizontal and vertical dimension of circuits.

Basically for every circuit, if it is rotated then cxr is cy and cyr is cx, and if it is not rotated then cxr is cx and cyr is cy.

I changed cx and cy in the constraints with cxr and cyr and added a constraint for when the circuit height is greater than the plate width it should not be rotated.

2.5 Validation

The baseline model that I used here uses gecode with system breaking. At the beginning I was getting subpar results, in that my model was not solving many instances although the ones being solved bore satisfying results. This was because I was letting my model restart its search using 4 different processes to find the most optimal solution. While it did find the most optimal solution for some, it was also too long a process to find the most optimal solution for others before exceeding the 5 minutes time limit.

Therefore, I opted out of restarting the search using these processes and only left the time limit for the mode. This acheieved not only the same optimal results but even better since the model had the opportunity to solve even more instances. I have included in Table 1 the results of the original model although I only utilized 20 instances.

After the baseline model I decided to remove the symmetry breaking constraints to see what these results would yield.

I then decided to run the models using Chuffed instead of gecode, although I only ran it with the normal model since it yeilded worse results. I wrote the code to allow a user-input that asked the user if they want to use gecode and

Chuffed. For the Chuffed version I kept the System Breaking constraints since it yeilded better results with gecode.

Refer to Table 1 and 2 for all the results of the heights found for each instance in all my models.

3 SMT Model

3.1 Decision variables

The decision variables are w, n, cx, cy, x, y, lower, upper, and h.

I equated the height to the lower bound to begin with and set a while loop that states that while the solution is not found and the height is less than the upper bound, the solver is going to keep looking for a solution. For every solution not found the height is incremented by one and the search repeats.

Refer to Section 1.1 for the rest of the decision variables.

3.2 Objective function

The Objective Function is to minimize the height.

Refer to Section 3.1 for how.

3.3 Constraints

The constraints in the code ensure that the pieces are placed within the plate without overlapping with each other, and that the height of the plate is minimized. The code also includes a time limit for the solver, to prevent it from running longer than 5 minutes.

3.3.1 Domain Reduction and No Overlap Constraints

This code is written in the context of an optimization problem, where the goal is to find the optimal placement of a set of circles on a rectangular grid.

1. Setting domain constraints: The domain constraints ensure that each circle's x and y coordinate must be within the grid. This is done by defining two lists ”domain x” and ”domain y”. For each circle (iterated over by the variable ”i”), the code appends constraints to these lists that ensure the x coordinate is non-negative (x[i] ¿= 0) and that the x coordinate plus the width of the circle (cx[i]) is less than or equal to the width of the grid (w). Similar constraints are applied to the y coordinate, ensuring that it is non-negative (y[i] ¿= 0) and that the y coordinate plus the height of the circle (cy[i]) is less than or equal to the height of the grid (h).

2. Setting no overlap constraints: The no overlap constraints ensure that no two circles can overlap each other. This is done by defining a list "no overlap". For each pair of circles (iterated over by the variables "i" and "j"), the code appends a constraint to this list that ensures the two circles do not overlap. This is done by using the Or function, which takes a list of conditions as input and returns True if any of the conditions are True. In this case, the conditions are that either the x coordinate of the first circle plus its width is less than or equal to the x coordinate of the second circle, or the x coordinate of the second circle plus its width is less than or equal to the x coordinate of the first circle, or the y coordinate of the first circle plus its height is less than or equal to the y coordinate of the second circle, or the y coordinate of the second circle plus its height is less than or equal to the y coordinate of the first circle.

Finally, the domain constraints and no overlap constraints are added to the optimization problem using the add() method, which takes a list of constraints as input.

3.4 Rotation

The lower and upper bounds given here for the rotation models are:

$$lowest = \max_{i=0}^{circ-1} \min(cx_i, cy_i) \tag{7}$$

$$lower = \max\left(lowest, \left\lceil \frac{\sum_{i=0}^{circ-1} cx_i \cdot cy_i}{w} \right\rceil\right) \tag{8}$$

$$upper = \left\lfloor \sum_{i=0}^{circ-1} \max(cy_i, cx_i) \right\rfloor \tag{9}$$

I also introduced a Boolean variable called rot that whether a circuit is rotated or not. isrot is True if it is rotated and False if not.

Furthermore, added two variables cxr and cyr that represents the actual horizontal and vertical dimension of circuits.

Basically for every circuit, if it is rotated then cxr is cy and cyr is cx, and if it is not rotated then cxr is cx and cyr is cy.

I also changed cx and cy in the constraints with cxr and cyr and added a constraint for if a circuit is squared, then force it to be not rotated.

3.5 Validation

My model was implemented using Z3 solver.

Refer to Table 3 for the results of the heights found for each instance in my normal and rotate models using Z3 solver.

4 MIP Model

4.1 Decision variables

The decision variables are w, n, cx, cy, x, y, lower, upper, and h.

Refer to Section 1.1

4.2 Objective function

The Objective Function is to minimize the height.

Refer to Section 2.2

4.3 Constraints

Same Constraints as Section 3.3

4.4 Rotation

Same changes as Section 3.4 except that rot is a binary variable not a Boolean variable that returns 1 if the circuit is rotate and 0 otherwise.

4.5 Validation

My model was implemented using Pulp. Refer to Table 4 for the results of the heights found for each instance in my normal and rotate models using Z3 solver.

5 Conclusion

We can conclude that while CP and SMT yeilded satisfying results, MIP was not as efficient in the number of how many instances it solved in the constraints given. And according to tables 1, 2, and 3, while SMT had the best results for the normal model, CP yeilded much better results for the rotation model. Refer to Table 5 and Table 6 to see the best optimizations of all three approaches.

Table 1: CP Normal

ID	CP-gecode-SB(N)(4P)	CP-gecode-SB(N)	CP-gecode-noSB(N)	CP-Chuffed-(N)
Ins 1	8	8	8	8
Ins 2	9	9	9	9
Ins 3	10	10	10	10
Ins 4	11	11	11	11
Ins 5	12	12	12	12
Ins 6	13	13	13	13
Ins 7	14	14	14	14
Ins 8	15	15	15	15
Ins 9	16	16	16	16
Ins 10	17	17	17	17
Ins 11	N/A	N/A	N/A	18
Ins 12	N/A	19	19	19
Ins 13	N/A	20	20	20
Ins 14	N/A	21	21	21
Ins 15	N/A	22	22	22
Ins 16	N/A	N/A	N/A	N/A
Ins 17	N/A	24	24	24
Ins 18	N/A	25	25	25
Ins 19	N/A	26	26	N/A
Ins 20	N/A	27	27	N/A
Ins 21		28	28	N/A
Ins 22		N/A	N/A	N/A
Ins 23		30	N/A	N/A
Ins 24		31	31	N/A
Ins 25		N/A	N/A	N/A
Ins 26		33	33	N/A
Ins 27		34	34	N/A
Ins 28		35	N/A	N/A
Ins 29		36	N/A	N/A
Ins 30		N/A	N/A	N/A
Ins 31		N/A	N/A	N/A
Ins 32		N/A	N/A	39
Ins 33		40	40	N/A
Ins 34		N/A	N/A	N/A
Ins 35		N/A	N/A	N/A
Ins 36		40	40	40
Ins 37		N/A	N/A	N/A
Ins 38		N/A	N/A	N/A
Ins 39		N/A	N/A	N/A
Ins 40		N/A	N/A	N/A
Total	**10/20**	**27**	**24**	**19**

Table 2: CP Rotate

ID	CP-gecode-SB(R)	CP-gecode-noSB(R)
Ins 1	8	8
Ins 2	9	9
Ins 3	10	10
Ins 4	11	11
Ins 5	12	12
Ins 6	13	13
Ins 7	14	14
Ins 8	15	15
Ins 9	16	16
Ins 10	17	17
Ins 11	18	18
Ins 12	19	19
Ins 13	20	20
Ins 14	21	21
Ins 1	22	22
Ins 16	N/A	N/A
Ins 17	24	24
Ins 18	N/A	N/A
Ins 19	N/A	N/A
Ins 20	N/A	N/A
Ins 21	28	N/A
Ins 22	N/A	N/A
Ins 23	N/A	N/A
Ins 24	31	31
Ins 25	N/A	N/A
Ins 26	N/A	N/A
Ins 27	34	34
Ins 28	35	N/A
Ins 29	N/A	N/A
Ins 30	N/A	N/A
Ins 31	N/A	N/A
Ins 32	N/A	N/A
Ins 33	40	40
Ins 34	N/A	N/A
Ins 35	40	40
Ins 36	40	40
Ins 37	N/A	N/A
Ins 38	N/A	N/A
Ins 39	N/A	N/A
Ins 40	N/A	N/A
Total	**23**	**21**

Table 3: SMT

ID	Z3 Normal	Z3 Rotate
Ins 1	8	8
Ins 2	9	9
Ins 3	10	10
Ins 4	11	11
Ins 5	12	12
Ins 6	13	13
Ins 7	14	14
Ins 8	15	15
Ins 9	16	16
Ins 10	17	17
Ins 11	18	N/A
Ins 12	19	19
Ins 13	20	20
Ins 14	21	21
Ins 15	22	22
Ins 16	23	N/A
Ins 17	24	N/A
Ins 18	25	N/A
Ins 19	26	N/A
Ins 20	27	N/A
Ins 21	28	N/A
Ins 22	29	N/A
Ins 23	30	N/A
Ins 24	31	N/A
Ins 25	32	N/A
Ins 26	33	N/A
Ins 27	34	N/A
Ins 28	35	N/A
Ins 29	36	N/A
Ins 30	N/A	N/A
Ins 31	38	38
Ins 32	N/A	N/A
Ins 33	40	N/A
Ins 34	N/A	N/A
Ins 35	40	N/A
Ins 36	40	40
Ins 37	N/A	N/A
Ins 38	N/A	N/A
Ins 39	N/A	N/A
Ins 40	N/A	N/A
Total	**33**	**16**

Table 4: MIP Model

ID	PULP Normal	PULP
Ins 1	8	8
Ins 2	9	9
Ins 3	10	10
Ins 4	11	11
Ins 5	12	12
Ins 6	13	13
Ins 7	14	14
Ins 8	15	15
Ins 9	16	N/A
Ins 10	17	N/A
Ins 11	N/A	N/A
Ins 12	N/A	N/A
Ins 13	N/A	N/A
Ins 14	N/A	N/A
Ins 15	N/A	N/A
Ins 16	N/A	N/A
Ins 17	N/A	N/A
Ins 18	N/A	N/A
Ins 19	N/A	N/A
Ins 20	N/A	N/A
Ins 21	N/A	N/A
Ins 22	N/A	N/A
Ins 23	N/A	N/A
Ins 24	N/A	N/A
Ins 25	N/A	N/A
Ins 26	N/A	N/A
Ins 27	N/A	N/A
Ins 28	N/A	N/A
Ins 29	N/A	N/A
Ins 30	N/A	N/A
Ins 31	N/A	N/A
Ins 32	N/A	N/A
Ins 33	N/A	N/A
Ins 34	N/A	N/A
Ins 35	N/A	N/A
Ins 36	N/A	N/A
Ins 37	N/A	N/A
Ins 38	N/A	N/A
Ins 39	N/A	N/A
Ins 40	N/A	N/A
Total	**10**	**8**

Table 5: Normal Models

ID	CP-gecode-SB	SMT	MIP
Ins 1	8	8	8
Ins 2	9	9	9
Ins 3	10	10	10
Ins 4	11	11	11
Ins 5	12	12	12
Ins 6	13	13	13
Ins 7	14	14	14
Ins 8	15	15	15
Ins 9	16	16	16
Ins 10	17	17	17
Ins 11	N/A	18	N/A
Ins 12	19	19	N/A
Ins 13	20	20	N/A
Ins 14	21	21	N/A
Ins 15	22	22	N/A
Ins 16	N/A	23	N/A
Ins 17	24	24	N/A
Ins 18	25	25	N/A
Ins 19	26	26	N/A
Ins 20	27	27	N/A
Ins 21	28	28	N/A
Ins 22	N/A	29	N/A
Ins 23	30	30	N/A
Ins 24	31	31	N/A
Ins 25	N/A	32	N/A
Ins 26	33	33	N/A
Ins 27	34	34	N/A
Ins 28	35	35	N/A
Ins 29	36	36	N/A
Ins 30	N/A	N/A	N/A
Ins 31	N/A	38	N/A
Ins 32	N/A	N/A	N/A
Ins 33	40	40	N/A
Ins 34	N/A	N/A	N/A
Ins 35	N/A	40	N/A
Ins 36	40	40	N/A
Ins 37	N/A	N/A	N/A
Ins 38	N/A	N/A	N/A
Ins 39	N/A	N/A	N/A
Ins 40	N/A	N/A	N/A
Total	**27**	**33**	**10**

Table 6: Normal Models

ID	CP-gecode-SB	SMT	MIP
Ins 1	8	8	8
Ins 2	9	9	9
Ins 3	10	10	10
Ins 4	11	11	11
Ins 5	12	12	12
Ins 6	13	13	13
Ins 7	14	14	14
Ins 8	15	15	15
Ins 9	16	16	N/A
Ins 10	17	17	N/A
Ins 11	18	N/A	N/A
Ins 12	19	19	N/A
Ins 13	20	20	N/A
Ins 14	21	21	N/A
Ins 15	22	22	N/A
Ins 16	N/A	N/A	N/A
Ins 17	24	N/A	N/A
Ins 18	N/A	N/A	N/A
Ins 19	N/A	N/A	N/A
Ins 20	N/A	N/A	N/A
Ins 21	28	N/A	N/A
Ins 22	N/A	N/A	N/A
Ins 23	N/A	N/A	N/A
Ins 24	31	N/A	N/A
Ins 25	N/A	N/A	N/A
Ins 26	N/A	N/A	N/A
Ins 27	34	N/A	N/A
Ins 28	35	N/A	N/A
Ins 29	N/A	N/A	N/A
Ins 30	N/A	N/A	N/A
Ins 31	N/A	38	N/A
Ins 32	N/A	N/A	N/A
Ins 33	40	N/A	N/A
Ins 34	N/A	N/A	N/A
Ins 35	40	N/A	N/A
Ins 36	40	40	N/A
Ins 37	N/A	N/A	N/A
Ins 38	N/A	N/A	N/A
Ins 39	N/A	N/A	N/A
Ins 40	N/A	N/A	N/A
Total	**23**	**16**	**8**