VLSI Design Problem

Enrico Morselli enrico.morselli@studio.unibo.it

January 2023

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. This report describe a Combinatorial Optimization approach to the VLSI Design problem, where given a fixed-width plate and a list of rectangular circuits, we have to decide how to place them on the plate so that the length of the final device is minimized. In particular, three different technologies are employed to address the problem at hand, namely Constraint Programming (CP), propositional SATisfiability (SAT) and Mixed-Integer Linear Programming (MIP). This project should have been done in a group of two people. Sadly, one of the two members quitted one month before the deadline to study for other exams, and i didn't find any other group so i tried working on the project alone. It has been quite challenging, expecially regarding the SAT section. Also, the writing of this report took a lot more time than expected.

1.1 Preliminaries

1.1.1 Format of the Instances

Each technique is applied to a set of provided instances of the VLSI Design Problem. In each instance we are given the width of the plate w, the number of circuits to place on the plate n and a set $D = \{(x_i, y_i) | i \in [1, \dots, n]\}$, where x_i and y_i are respectively the width and the height of the i-th circuit. Given these parameters, a solution involves determining the minimum feasible height h, as well as the locations of each of the n circuits, determined by $\hat{x_i}$ and $\hat{y_i}$, which are respectively the horizontal and vertical coordinate of the bottom left corner of the i-th circuit.

1.1.2 Pre-processing steps

In this section we define the common pre-processing steps for all technologies. First, we have to define a proper domain for the objective variable h, so we will define a lower bound h_{min} and an upper bound h_{max} based on the available knowledge. An estimation for the lower bound can be computed as:

$$h_{\text{min}} = \frac{\sum_{i=1}^{n}(x_i \cdot y_i)}{w}$$

i.e. summing the areas $(x_i \cdot y_i)$of each circuit i and dividing the result by the width w of the plate. This value for the height is actually feasible if and only if there exists a solution where we can pack all the given circuits in the plate without leaving any empty space between them. As an estimation for the upper bound h_{max} we simply use

$$h_{\text{max}} = \frac{\sum_{i=1}^{n}}{y_i}$$

i.e. the sum of the heights y_i of all circuits $i \in [1, \dots, n]$. That's because in theory we could always find a solution by stacking all circuits one on top of the other. The second pre-processing step involves sorting the circuits in order of increasing area, i.e. the first pair of coordinates (x_1, y_1) will correspond to the biggest circuit, coordinates (x_2, y_2) to the second biggest and so on.

2 CP Model

Constraint Programming is a paradigm for solving combinatorial problems by stating, in a declarative fashion, a set of constraints that must hold on the feasible solutions for a given set of decision variables. In this section, a CP model for the VLSI Design Problem is described.

2.1 Variables

In order to model the problem, we define the following parameters:

- w: The width of the plate;
- n: The number of circuits to be placed on the plate;
- $h_{\text{min}}, h_{\text{max}}$: Lower and upper bound for the plate height;
- $\mathbf{x}$: An array of integers, representing the horizontal dimensions of the n circuits;
- $\mathbf{y}$: An array of integers, representing the vertical dimensions of the n circuits.

Then, we define the following decision variables:

- h: an integer representing the height of the plate. The domain of this variable is given by $[h_{\min}, h_{\max}]$, where $h_{\min}$ and $h_{\max}$ are respectively the lower and upper bound defined in section 1.1.2.

- $\hat{\mathbf{x}}$: an array of integers representing the placement of the circuits inside the plate with respect to the horizontal dimension, as defined in section 1.1.1. The domain of $\hat{x}$ is defined as $[0, w - x_{\min}]$, where w is the width of the plate and $x_{\min}$ is the minimum of $\mathbf{x}$, i.e. the smallest between all the horizontal circuit dimensions. This encodes the fact that each circuit must be placed only inside the borders of the plate.

- $\hat{\mathbf{y}}$: an array of integers representing the placement of the circuits inside the plate with respect to the vertical dimension, as defined in section 1.1.1. The domain of $\hat{y}$ is defined as $[0, h_{\max} - y_{\min}]$, where $y_{\min}$ is the minimum element of $\mathbf{y}$.

2.2 Objective function

Our objective for this problem is to minimize the height h as defined in the previous section. We have already defined an upper and lower bound for h in section 1.1.2 so we don't need further considerations.

2.3 Constraints

2.3.1 Main Problem Contraints

First we need to specify that circuits should not overlap with one another, i.e. each position inside the board must be occupied by at most one circuit. The global constraint library of MiniZinc offers the `diffn` constraint, which serves exactly for this purpose. More precisely, the constraint `diffn(x, y, dx, dy)` states that the rectangles with origin `(x, y)` and dimensions `(dx, dy)` should not overlap. So we simply use this constraint with $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ in place of `(x, y)` and $(\mathbf{x}, \mathbf{y})$ in place of `(dx, dy)`. In general, whenever we find a global constraint which does exactly what we want, the best option is to use that, since many solvers implements optimized propagation algorithms for such global constraints, thus making the search process faster. Then, we need to give bounds to the origin of each circuit. More precisely, for each circuit i, the horizontal coordinate $\hat{x_i}$ must be less than or equal than the width of the plate minus the horizontal dimension of the circuit, i.e.

$$\forall i \in [1, n], \hat{x_i} \leq w - x_i$$

The same holds for the vertical coordinate of each circuit, which must be less than or equal the maximum height of the plate minus the height of the circuit

$$\forall i \in [1, n], \hat{y_i} \leq h_{\max} - y_i$$

2.3.2 Implied Constraints

Implied constraints are logical consequences of the initial specification of the problem. Even though they do not change the set of solutions they are useful because they increase the amount of propagation and thus reduce the amount of search the solver has to do. We can think of implied constraints like a dead end sign at the start of a road: if such sign is not present, we will traverse the road until the end, and then we will be forced to go back (backtrack) and thus wasting time. In the case of this problem, we can derive two implied constraints by seeing the VLSI Design problem as a cumulative resource problem. In particular, we consider the plate as a shared resource, the height h of the board as the resource capacity. Then, the circuits $i = 1, \ldots, n$ are tasks with starting time $\hat{x}$ and duration x, which can be assigned simultaneously to the shared resource (i.e. stacked on top of the other inside the board) and at any time they must not exceed the total resource capacity. In other words, at any point $\hat{x}$, the sum of all the horizontal dimensions $y_1, y_2, \ldots$ must be less than or equal to the height h. We can express this by using the global constraint `cumulative(s, d, r, b)`, which 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. So we add the constraint `cumulative(xhat, x, y, h)` (where `xhat` corresponds to $\hat{x}$. We can do the exact same reasoning in the other dimension, i.e. at any point along the vertical axis, the sum of the horizontal dimensions of the circuits should be less than or equal to the width of the plate. Then we can post the constraint `cumulative(yhat, y, x, w)`. These two constraints are implied because from the main problem constraints we can derive that $\forall i \in [1, n], \hat{x_i} + x_i \leq w$ and $\forall i \in [1, n], \hat{y_i} + y_i \leq h$, and thus the two `cumulative` constraints are a logical consequence of the two main problem constraints.

2.3.3 Symmetry breaking constraints

Rotation and reflection. In the VLSI Design Problem we can identify many symmetries. Given a solution, we can find a symmetric solution by rotating the plate by 90°, 180° or 270°. Also, we could find a symmetric solution by flipping the plane along the horizontal axis or the vertical axis or both of them. To break such symmetry we impose an ordering between the two largest circuits, forcing the biggest circuit to be at the bottom left of the second biggest.

Row and Column Block symmetry. Whenever two items i, j of equal height are besides each other, they can be freely swapped, because the combined size of the two item will not change in the final solution. The same applies for items of equal width on top of each other. To break this type of symmetry, we impose an ordering between the two items on the varying coordinate whenever we have this kind of situation.

Three-Block symmetry. This symmetry involves 2 adjacent circuits i, j with the same y-dimension $y_i = y_j$, which are both adjacent to a bigger circuit k for which we have $x_i + x_j = xk$. In this case, we can freely swap the "composite" circuit i, j with circuit k. A similar reasoning applies to the horizontal case. To break this symmetries, we impose an ordering on the varying coordinate of i and k.

2.4 Rotation

The general case of the VLSI Design problem allows the rotation of circuits, meaning that a circuit with dimensions $x_k \times y_k$ can be positioned on the plate inverting its height and width, so as $y_k \times x_k$. In order to handle rotation in the model, additional variables and constraints are added. In particular, we introduce a boolean array with size n, named `rotated`, such that for $i = 1, \dots, n$, $rotated(i) = True$ if circuit i is rotated with respect to its original dimensions and false otherwise. We also insert two new arrays x_{input} and y_{input} to store the original input dimensions from the instance file, so the domain will be the same as the two original `x` and `y` arrays. These two will in turn be converted into decision variables, because the value of x_i and y_i depends on whether circuit i is rotated or not. We introduce the following constraint to invert the dimensions of a circuit in case its rotated.

$$rotated(i) \Rightarrow (x_i = input_y) \wedge (y_i = input_x) \quad \forall i \in [1, n]$$

Lastly, a constraint is imposed to prevent square circuits from rotating, since their sizes would remain the same:

$$(input_x = input_y) \Rightarrow \neg rotated(i) \quad \forall i \in [1, n]$$

2.5 Validation

2.5.1 Experimental design

TODO: maybe add more details for VOHs and Restart strategies (?) One of the key features of CP is the possibility of interacting with the search process by specifying different search strategies, such as heuristics for variable and value ordering, and restart strategies. The search process in MiniZinc can be customized through annotations, which allows for specifying the search strategy, including variable and value ordering heuristics, and restart strategies. As variable ordering heuristcs (VOHs) i tried using:

- `input_order`: chooses the variables in the given order, which for our model means to choose the biggest circuit not yet positioned (we order the input items by area in non increasing order, as explained in Section 1.1.2).
- `first_fail`: chooses the variable x with minimum domain size (TODO: explain)

- `dom_w_deg`: chooses the variable x that minimizes $dom(x)/w(x)$, where $dom(x)$ is the (current) domain size of x and $w(x)$ is the weighted degree of x, computed as the sum of the weights of all constraints involving x (initially set to 1 and incremented each time the constraint fails).

As value ordering heuristic for `input_order` and `first_fail` i used `indomain_min`, which chooses the minimum value in the domain of the variable. For `dom_w_deg` instead, i tried using `indomain_random`, since `dom_w_deg` gives good results when paired with randomization.
Then, i also added the possibility of specifying a restart strategy to the solver, to control how often a restart will occur. We consider the following restart strategies (TODO:explain better maybe):

- `restart_none`: no restart strategy is applied.
- `restart_luby(scale)`: the i-th restart occurs after $L[i] \cdot$ scale nodes in the search tree, where $L[i]$ is the i-th number of the Luby sequence. We set scale $= 150$.
- `restart geometric(base, scale)`: the i-th restart occurs after scale $\cdot$ basei nodes. We set base $= 2$ and scale $= 50$.

By the way, it makes sense to use restart strategies when using randomization, since it won't make sense to restart if the search tree is going to be visited in the same order every time. TODO: explain better.
In order to launch the search i defined a Python script `solve_cp.py` with the following usage:

```
solve_cp.py [-h] [-s START] [-e END] [-t TIMEOUT] [-r] [-sb]
                 [--solver SOLVER] [--heu HEU] [--restart RESTART]
```

Here are the parameters that can be specified

- `-s`: number of the first instance to solve;
- `-e`: number of the first instance to solve;
- `-t`: number of seconds before time out. By default, we do not consider instances which takes more than 5 minutes, so the default timeout is 300 seconds;
- `-r`: allows the rotation of the circuits;
- `-sb`: allows the use of symmetry breaking constraints;
- `--solver`: solver to use;
- `--heu`: variable ordering heuristic to use;
- `--restart`: restart strategy to use.

Hardware Specifications. All the experiments were run on a laptop with the following specifications

- MacBook Pro, 13-inch, 2019;
- CPU: 1,4 GHz Quad-Core Intel Core i5;
- RAM: 8 GB 2133 MHz LPDDR3;
- OS: macOS Ventura 13.0.1

Before presenting your experimental results, give all the details of your experimental study. Your results should be reproducible following your description. Explain which solvers you used and which search strategies you employed, as well as your experimental set up (e.g., the hardware and the software used, any posed time limit etc). Irreproducible experiments will not be considered.

2.5.2 Experimental results

We present the experimental results of this model in the following tables. The first one shows the results of the first variant of the problem, while the second shows the results of the model with rotation allowed.

Present your experimental results in a clear way. It is mandatory to show a table where:

- rows are labeled with the identifier of the instances,

ID	Chuffed + SB	Chuffed w/out SB	Gecode + SB	Gecode w/out SB
1	100	120	**80**	80
2	**50**	60	N/A	N/A
3	UNSAT	UNSAT	N/A	N/A
4	N/A	N/A	N/A	N/A
5	N/A	N/A	N/A	N/A
6	N/A	N/A	N/A	N/A
7	N/A	N/A	N/A	N/A
8	N/A	N/A	N/A	N/A
9	N/A	N/A	N/A	N/A
10	N/A	N/A	N/A	N/A
11	N/A	N/A	N/A	N/A
12	N/A	N/A	N/A	N/A
13	N/A	N/A	N/A	N/A
14	N/A	N/A	N/A	N/A
15	N/A	N/A	N/A	N/A
16	N/A	N/A	N/A	N/A
17	N/A	N/A	N/A	N/A
18	N/A	N/A	N/A	N/A
19	N/A	N/A	N/A	N/A
20	N/A	N/A	N/A	N/A
21	N/A	N/A	N/A	N/A
22	N/A	N/A	N/A	N/A
23	N/A	N/A	N/A	N/A
24	N/A	N/A	N/A	N/A
25	N/A	N/A	N/A	N/A
26	N/A	N/A	N/A	N/A
27	N/A	N/A	N/A	N/A
28	N/A	N/A	N/A	N/A
29	N/A	N/A	N/A	N/A
30	N/A	N/A	N/A	N/A
31	N/A	N/A	N/A	N/A
32	N/A	N/A	N/A	N/A
33	N/A	N/A	N/A	N/A
34	N/A	N/A	N/A	N/A
35	N/A	N/A	N/A	N/A
36	N/A	N/A	N/A	N/A
37	N/A	N/A	N/A	N/A
38	N/A	N/A	N/A	N/A
39	N/A	N/A	N/A	N/A
40	N/A	N/A	N/A	N/A

Table 1: Results using Gecode and Chuffed with and without symmetry breaking using the search strategy h.

- columns are labeled with the different approaches you tried,

- cells contain the best objective value (not the runtime), as specified in the project description, found by the given approach on the given instance, using a certain search strategy. If the instance is solved to optimality, emphasise the objective value in bold. If the instance is proved to be unsatisfiable, indicate it as 'UNSAT'. If no answer is obtained within the time limit, indicate it with a 'N/A' or '-'.

For example, "The results obtained using the search strategy h are reported in Table 1". The runtimes can be depicted using plots.

3 SAT Model

The Boolean Satisfiability problem (SAT) involves determining whether a given propositional formula has a satisfying assignment of its variables, i.e. whether there is an interpretation which makes the formula evaluate to true. The goal is to find an efficient SAT encoding for the VLSI problem, which then can be passed to a SAT solver. After a bit of research i decided to follow the approach suggested in the paper by T. Soh et. al. (TODO: insert bibliography and references) for the 2SPP. Since using SAT solvers we cannot define a target function to optimize (which in this case would be the height of the plate), to find the optimal height, we will solve a sequence of sub-problems of the original VLSI, where the height of the plate h is fixed. In practice, we will start from the lower bound $h_{\min}$ defined in Section 1.1.2, and check each possible value until we find the minimum height for which the problem is SAT (i.e. the optimal height).

3.1 Order Encoding

To encode the problem in SAT Soh. et. al. used the so called Order encoding of variables: Let x be an integer variable of domain $\{0, \dots, n\}$, and c an integer value with domain $\{0, \dots, n-1\}$. At the first step, a constraint with comparison is translated into primitive comparisons which are in the form $x \leq c$. At the next step, a primitive comparison is encoded into a Boolean variable $p_{x,c}$. The value of x can therefore be encoded in the set of literals $\{px_0, \dots, px_{n1}\}$ of size n. An example of order encoding can be found in the following table:

value of x	order encoding
0	T, T, T
1	F, T, T
2	F, F, T
3	F, F, F

As we may notice from the example, with order encoding we never have a $False$ value after a $True$ value. If for instance we have an interpretation where $x = 1$, the clause $x \leq 0$ does not hold (i.e. $px_0 = False$), but instead $x \leq 1$ does (i.e. $px_1 = True$. This is holds also for all comparisons $x \leq c$ where $c \geq 1$ (i.e. all the successive clauses px_c are True). Therefore, it never happens that px_c is True and px_{c+1} is False. Then, in a SAT encoding we need the following clause:

$$px_c \implies px_{c+1} \quad c \in \{0, \ldots, n-2\}$$

Which can be rewritten in CNF form as:

$$\neg px_c \vee px_{c+1} \quad c \in \{0, \ldots, n-2\}$$

In the following, we will refer to such clauses as ordering constraints.

3.2 Comparing Integers.

We then need a way to encode comparisons between integers, For example, given two integer variables $x_1, x_2 \in \{0, \ldots, n\}$ and an integer constant a, we need a way to transform the comparison $x_1 + a \leq x_2$ in a CNF on the literals $px_{1,c}$ and $px_{2,c}$, with $cin\{0, \ldots, n-1\}$. In order encoding, this is achieved by writing the comparison as a CNF of primitive comparisons $x_i \leq c$ for each value of c. This then directly corresponds to the literal $px_{i,c}$. To do so, we first need to reduce the domains of x_1 and x_2 to make sure that $x_1 \leq na$ and $x_2 \geq a$. We need to add the following primitive comparisons as constraints:

$$\begin{array}{ll} x_1 \leq k & k \in \{n-a, \ldots, n\} \\ x_2 \geq k & k \in \{0, \ldots, a\} \end{array}$$

For x_2, we need to transform the $\geq$ in $\leq$. We therefore obtain:

$$\neg(x_2 \leq k) \quad k \in \{0, \ldots, a-1\}$$

Finally, we have to add the implications for all the possible intermediate values:

$$x_2 \leq a + k \implies x_1 \leq k \quad k \in \{0, \ldots, n-a-1\}$$

The final conjunction of primitive comparisons therefore looks like this:

$$\begin{array}{ll} x_1 \leq k & k \in \{n-a, \ldots, n\} \\ \neg(x_2 \leq k) & k \in \{0, \ldots, a-1\} \\ x_2 \leq a + k \implies x_1 \leq k & k \in \{0, \ldots, n-a-1\} \end{array}$$

Since each comparison $x_i \leq c$ corresponds to the literal $px_{i,c}$, we obtain the final SAT encoding of the initial comparison as:

$$\begin{array}{ll} px_{1,k} & k \in \{n-a, \ldots, n-1\} \\ \neg px_{2,k} & k \in \{0, \ldots, a-1\} \\ \neg px_{2,a+k} \vee px_{1,k} & k \in \{0, \ldots, n-a-1\} \end{array}$$

3.3 Decision variables

Let $i, j \in [1, \ldots, n]$ be two circuits, with $i \neq j$, and e, f be any integer. Then the SAT encoding for our problem uses four kind of boolean atoms:

- $lr_{i,j}$, which is true if rectangle i is placed at the right of rectangle j;
- $ud_{i,j}$, which is true if rectangle i is placed at the downward of rectangle j;
- $px_{i,e}$, which is true if rectangle i is placed at less than or equal e;
- $py_{i,f}$, which is true if rectangle i is placed at less than or equal f;

Then, for the purpose of searching for the optimal height, we added a fifth set of variables ph, such that ph_i is true if all circuits are placed below height h_i. The following section explains how this variables were used.

3.4 Objective function

As already mentioned in Section 3, we cannot define a target function to minimize. Then, we start by adding the constraint $ph_{h_{\min}}$, which is true if all circuits are packed under $h_{\min}$, where $h_{\min}$ is our lower bound for the plate height defined in 1.1.2, and we check for satisfiability. If we get UNSAT, we remove $ph_{h_{\min}}$ and we add $ph_{h_{\min}+1}$, and we check for satisfiability again. We repeat this process until we find an height for which the problem is SAT, which means we have found the optimal height.

3.5 Constraints

Ordering Constraints. Given a plate of width , for each circuit i of dimensions w_i, h_i, we have the 2-literal axiom clauses due to order encoding:

$$\begin{array}{ll} \neg px_{i,e} \lor px_{i,e+1} & e \in \{0, \ldots, W-1\} \\ \neg py_{i,f} \lor py_{i,f+1} & f \in \{0, \ldots, H-1\} \end{array}$$

We also need to reduce the domain of the possible positions, because a rectangle i (of width w_i) must be positioned such that $x_i \leq W w_i$. If we do not impose this constraint, the rectangle's right border would end up further right than the rightmost edge of the strip. The same holds true for the y coordinate. We therefore add the domain-reducing constraints:

$$\begin{array}{ll} px_{i,e} & e \in \{W - w_i, \ldots, W-1\} \\ py_{i,f} & f \in \{H - h_i, \ldots, H-1\} \end{array}$$

Non overlapping constraints. For each pair of circuits i and j, with $i < j$), we have the following four literal clause as non-overlapping constraint.

$$lr_{i,j} \lor lr_{j,i} \lor ud_{i,j} \lor ud_{j,i}$$

The meaning of this clause is that rectangle j must not be positioned inside rectangle i. However, this clause alone is not sufficient, because it does not take into account the width and height of rectangle j. We therefore need additional clauses. For instance, if we position rectangle i left of rectangle j, we have to make sure that there is no overlap, i.e. that $x_i + w_i \leq x_j$. The same holds for the y coordinate, and when we invert the roles of rectangles i and j. This can be summarized by the following clauses:

$$\begin{aligned} lr_{i,j} &\implies x_i + w_i \leq x_j \\ lr_{j,i} &\implies x_j + w_j \leq x_i \\ ud_{i,j} &\implies y_i + h_i \leq y_j \\ ud_{j,i} &\implies y_j + h_j \leq y_i \end{aligned}$$

For each pair of circuits i and j, with $i < j$). In section 3.2 we have seen how to encode such comparisons. As an example, we will show how the first clause will be encoded (the other ones are analogous):

$$\begin{array}{lll} lr_{i,j} \implies px_{i,e} & & e \in \{W - w_i, \ldots, W - 1\} \\ lr_{i,j} \implies \neg px_{j,e} & & e \in \{0, \ldots, w_i - 1\} \\ lr_{i,j} \implies px_{j,e+w_i} & \implies px_{i,e} & e \in \{0, \ldots, W - w_i - 1\} \end{array}$$

Which can be converted into disjunctions as follows (the first one can be omitted because it is redundant with the domain reducting constraint):

$$\begin{array}{ll} \neg lr_{i,j} \vee \neg px_{j,e} & e \in \{0, \ldots, w_i - 1\} \\ \neg lr_{i,j} \vee \neg px_{j,e+w_i} \vee px_{i,e} & e \in \{0, \ldots, W - w_i - 1\} \end{array}$$

3.6 Symmetry Breaking Constraints

We implemented three different Symmetry Breaking constraints described by Soh et. al.

Large Rectangle. Reduces the possibility for placing large rectangles. The idea is that if, for each two rectangles i and j, if $w_i + w_j > W$, they cannot be placed alongside each other, so it is always the case that $ud_{i,j}$ or $ud_{j,i}$ is true. Therefore, if that is the case, we can remove from the overlapping constraints all the clauses containing $lr_{i,j}$ and $lr_{j,i}$.

Same Rectangles. for each two rectangles i and j, if they have equal sizes, i.e. $w_i = w_j$ and $h_i = h_j$, they can be freely swapped without changing their total area. Then, we can break these symmetries by imposing an ordering between the two rectangles. In this case, we modify the non overlapping constraints by eliminating those containing $lr_{j,i}$, thus forcing rectangle i to be placed at the left of rectangle j.

Largest Rectangle. We can cut some symmetric solutions also by reducing the domain of the largest circuit m, by imposing $x_m \leq \lfloor \frac{W-w_m}{2} \rfloor$ and $y_m \leq \lfloor \frac{H-h_m}{2} \rfloor$, to make sure that m is placed on the bottom left part of the grid. Therefore, we modify the domain reducing constraints for$m$:

$$\begin{array}{ll} px_{i,e} & e \in \{\lfloor \frac{W-w_m}{2} \rfloor, \dots, W-1\} \\ py_{i,f} & f \in \{\lfloor \frac{H-h_m}{2} \rfloor, \dots, H-1\} \end{array}$$

Then, if a circuit i has a width $w_i \geq \lfloor \frac{W-w_m}{2} \rfloor$, we must also modify the non-overlapping constraints to make sure that it cannot be placed to the left of m, so we eliminate the clauses containing $lr_{i,m}$.

3.7 Rotation

In order to handle rotation, we add a new variable R_i, for each circuit i, such that R_i is $True$ if i is rotated and $False$ otherwise. Then, we already know that if we rotate i its dimensions w_i and h_i will be inverted. Then we need to modify the domain-reducing constraints, by adding implications from R_i or $negR_i$:

$$\begin{array}{rcll} \neg R_i & \implies & px_{i,e} & e \in \{W-w_i, \dots, W-1\} \\ \neg R_i & \implies & py_{i,f} & f \in \{H-h_i, \dots, H-1\} \end{array}$$

$$\begin{array}{rcll} R_i & \implies & px_{i,e} & e \in \{W-h_i, \dots, W-1\} \\ R_i & \implies & py_{i,f} & f \in \{H-w_i, \dots, H-1\} \end{array}$$

Then, also the non overlapping constraints are updated. The following is how the constraints involving $lr_{i,j}$ are modified (the other ones are modified accordingly)

$$\begin{array}{rcll} \neg R_i & \implies & \neg lr_{i,j} \vee \neg px_{j,e} & e \in \{0, \dots, w_i - 1\} \\ \neg R_i & \implies & \neg lr_{i,j} \vee \neg px_{j,e+w_i} \vee px_{i,e} & e \in \{0, \dots, W - w_i - 1\} \end{array}$$

$$\begin{array}{rcll} R_i & \implies & \neg lr_{i,j} \vee \neg px_{j,e} & e \in \{0, \dots, h_i - 1\} \\ R_i & \implies & \neg lr_{i,j} \vee \neg px_{j,e+w_i} \vee px_{i,e} & e \in \{0, \dots, W - h_i - 1\} \end{array}$$

3.8 Validation

The model was implemented with python and solved using the z3 solver. As for CP, i defined a python script with the following usage:

Where is possible to specify the following parameters:

-

4 MIP Model

In MIP (Mixed Integer Programming) a problem is defined as a set of real valued or integer valued variables, constraints on these variables are expressed as linear (or sometimes quadratic) equalities or inequalities. In the following, a MIP encoding for the VLSI problem is described.

4.1 Decision variables

As for the CP version (see Section 2.1), our decision variables are:

- h: an integer representing the height of the plate, with domain$[h_{\text{min}}, h_{\text{max}}]$;
- $\hat{\mathbf{x}}$: an array of integers representing the placement of the circuits inside the plate with respect to the horizontal dimension, with domain $[0, w - x_{\text{min}}]$;
- $\hat{\mathbf{y}}$: an array of integers representing the placement of the circuits inside the plate with respect to the vertical dimension, with domain $[0, h_{\text{max}} - y_{\text{min}}]$.

4.2 Objective function

As in Section 2.2, the objective of our model is to minimize the height h of the plate, which is subject to lower bound h_{min} and upper bound h_{max} as defined in 1.1.2

4.3 Constraints

As for section (TODO:insert ref), we define bounding constraints for the placement of the circuits, such that, $\forall i \in [1, n]$:

$$\begin{gathered} x_i + w_i \leq w \\ y_i + h_i \leq h \end{gathered}$$

To make sure circuits are placed within the border of the plate. We also pose a bounding constraint:

$$y_i + h_i \leq h_max$$

Then we add non-overlapping constraints, by use of the big-M trick:

$$\begin{gathered} \hat{x_i} + w_i \leq \hat{x_j} + M(1 - lr_{i,j}) \\ \hat{x_j} + w_i \leq \hat{x_i} + M(1 - lr_{j,i}) \\ \hat{y_i} + h_i \leq \hat{y_j} + M(1 - ud_{i,j}) \\ \hat{y_j} + h_i \leq \hat{y_i} + M(1 - ud_{j,j}) \end{gathered}$$

where $lr_{i,j}, lr_{j,i}, ud_{i,j}, ud_{j,j}$ are binary indicator variables which encodes the positions between two circuits i and j (similarly to the boolean atoms defined for SAT encoding in Section 3.3). The essence of the big-M trick is that here whenever one of such atoms is $False$, the M constant, which is an arbitrary large number will make the inequality $\geq$ always true. Conversely, if one of the indicators is $True$, the solver will be forced to find values for the variables such that the inequalities are satisfied. Then i put together the indicator variables in a 4-way OR, by forcing their sum to be ≥ 1:

$$lr_{i,j} + lr_{j,i} + ud_{i,j} + ud_{j,j} \geq 1$$

ID	CP	SMT	MIP
1	**80**	80	130
2	50	N/A	**50**
3	360	400	350

Table 2: The best result obtained for each instance.

4.4 Rotation

We handle rotation similarly as in Section 2.4, we add a new variable **rot** such that $rot_i = True$ if circuit i is rotated. Then, due to the inversion of sizes, we define two new decision variables **x** and **y**, which are integer arrays of size n, and we will name the original coordinates given by the instance as

See Section 2.5. Bonus points will be considered if a solver-independent language (e.g., $AMPL$) is employed so as to play with different MIP solvers on the same model.

5 Conclusions

Write down briefly your concluding remarks and show a final table with a row for each instance and a column for each optimization technology you considered. Each cell will contain the best objective value found by the given technology on the given instance (e.g., see Table 2).

References

[1] Donald E. Knuth (1986) *The TEX Book*, Addison-Wesley Professional.

[2] Leslie Lamport (1994) *LATEX: a document preparation system*, Addison Wesley, Massachusetts, 2nd ed.