VLSI Optimization

Klein Tahiraj, Vairo Di Pasquale
{*klein.tahiraj, vairo.dipasquale*}*@studio.unibo.it*

February 2, 2023

1 Introduction

The project aims to tackle Very Large-Scale Integration (VLSI), which is the process of packing circuits as efficiently as possible on a board with the purpose of creating a space-efficient chip. VLSI is a common process used to design a broad class of electronic devices and -lucky for us- is an interesting optimization problem. In the scope of this project, VLSI is approached using three different strategies, namely through constraint programming (CP), boolean satisfiability (SAT), and mixed integer programming (MIP). The input is always provided as a file specifying in order:

- the width of the board as `plate_width`;
- the total number of circuits, as `tot_circuits`;
- a list of the circuits widths, as `circuits_width`;
- and finally, a list of circuit heights, as `circuits_height`.

To ease the mathematical description of our models, we define here several entities:

$$p := \texttt{plate\_width} \tag{1}$$

$$C = \{1, ..., \texttt{tot\_circuits}\} \tag{2}$$

$$W = \{w_1, ..., w_{|C|} : w_i = \texttt{circuits\_width[i-1]}\} \tag{3}$$

$$H = \{h_1, ..., h_{|C|} : h_i = \texttt{circuits\_height[i-1]}\} \tag{4}$$

$$D_c = (w_c, h_c), w_c \in W, h_c \in H \tag{5}$$

$$D = \{D_1, ..., D_{|C|}\} \tag{6}$$

$$\mathbb{B} = \{True, False\} \tag{7}$$

$$\mathbb{Z}_+ = \{z \in \mathbb{Z} : z \geq 0\} \tag{8}$$

In all models the problem consists in placing each circuit D_c on an integer lattice σ of fixed width and infinite height:

$$\sigma : \mathbb{Z} \cap \{0, ..., p\} \times \mathbb{Z}_+ \tag{9}$$

Along all this report the subscript c, as in D_c will always represent $c \in C$.

Tables are shown at the end of this document.

2 CP Model

Constraint Programming involves formulating a problem as a set of constraints on a set of variables. W, H are stored in two integer arrays of dimension $|C|$, p and $|C|$ as integer values.

2.1 Decision variables

With the bottom left corner of our board placed on the origin of σ, we use as decision variables the coordinates of the bottom left corner of each circuit $c \in C$, namely x_c, y_c. In the code, these variables are declared as two vectors of variable values:

$$x = (x_1, ..., x_{|C|}) \in \mathbb{Z}_+^{|C|} \qquad y = (y_1, ..., y_{|C|}) \in \mathbb{Z}_+^{|C|} \tag{10}$$

In order to reduce the search space, we properly bound the domain of each coordinate. In the case of x coordinates, would not be convenient to choose p as an upper bound. Indeed, only the circuits with the smallest width could assume the upper bound of x as their abscissa, thus we limit it to p minus the width of the smallest circuit. A similar reasoning is followed for the y coordinates, with the best choice for theirs upper bound being the value of the height of the board minus the smallest of the heights among the circuits. For the mathematical description of this model, we define the position of a circuit c as a tuple

$$P_c = (x_c, y_c) \tag{11}$$

with

$$x_c \in [0, p - \min_{c \in C}(w_c)] \subset \mathbb{Z} \tag{12}$$

$$y_c \in [0, \eta - \min_{c \in C}(h_c))] \subset \mathbb{Z} \tag{13}$$

where η is a value indicating a fixed value for the board, which will be defined in more detail in the following section. To ease further description, we define the set of positions as:

$$P = \{P_1, ..., P_{|C|}\} \tag{14}$$

2.2 Objective Function

The most straightforward choice for the objective function is the maximum height occupied by the circuits once they are packed on the board. We call this variable η, and we declared it in code as a variable integer value. In order to further reduce the search space, also the codomain of η has been limited. Concerning the upper bound, We set it to the sum of the heights of all the circuits:

$$h^{top} = \sum_{c \in C} h_c \tag{15}$$

Although this is a limited case, the expressive power of MiniZInc was not enough to propose a more refined esteem, such as one that would also involve p and W. The choice for the lower bound is made by comparing two different esteems. The first esteem comes from the ideal case of circuits packed into a quadrilateral shape. Thus, we calculate an intrinsic minimum height as the sum of circuits areas divided by the width of the board, rounded to the smallest greater than or equal integer, namely:

$$h^{int} = \left\lceil \left(\sum_{c \in C} w_c h_c \right) p^{-1} \right\rceil \tag{16}$$

The second esteem accounts for the scenario of a circuit being taller than h^{int}. To deal with this case, we take the maximum value of height among the circuits as the other possible lower bound. In symbols:

$$h^{tall} = \max_{c \in C}(h_c) \tag{17}$$

Then, we take the biggest among the two esteems for the lower bound. In conclusion, we can define our objective function η as:

$$\eta(D, P) \colon \mathbb{Z}^{4|C|} \longrightarrow [max(h^{int}, h^{tall}), h^{top}] \subset \mathbb{Z} \tag{18}$$

The ultimate goal of our solver is to minimize η, i.e. to find an assignment of values for P such that, given D, the value returned by η is minimum. In order to model the problem, values of P must also satisfy some constraints.

2.3 Constraints

We implemented two classes of constraints: first some to model the problem and ensure that a solution is found, then some symmetry-breaking ones to speed up the computation. In this section, we'll refer to η as the value of height in which the circuits are constrained to be placed.

Modelling constraints

First, we ensure that each circuit is inside the board:

$$(x_c + w_c \leq p) \wedge (y_c + h_c \leq \eta) \quad \forall c \in C \tag{19}$$

Then, we ensure that circuits do not overlap by using the handy `diffn` native global function of Minizinc, borrowed from task scheduling:

$$\texttt{diffn}(x, y, W, H) \tag{20}$$

Also `diffn_nonstrict`, a variation of `diffn` which would allow for zero width circuits to be placed anywhere, was take into consideration, but because of the scope of the problem, it was judged unnecessary.

Finally, we ensure that circuits are properly distributed across the available area using `cumulative`, another native global constraint function of Minizinc, along both spatial dimensions:

$$\texttt{cumulative}(x, W, H, \eta); \tag{21}$$

$$\texttt{cumulative}(y, H, W, p) \tag{22}$$

This constraint is borrowed from task scheduling and enforces a set of tasks to have a specific usage resource over time, in this case, it ensures that circuits are spread along the area of the board and do not exceed limits. For example, the first ensures that the horizontal positions are such that the heights of the circuits do not exceed η.

Symmetry breaking constraints

First, we ensure that the bottom left corner of the circuit with the biggest area on the origin of our plane, namely:

$$x_k = 0 \land y_k = 0, k \in C : w_k h_k = \underset{c \in C}{max}(w_c h_c) \tag{23}$$

this breaks symmetries along the horizontal and vertical axes of the plane, as well as those along the $y = -x$ axis.

In the eventuality of the circuit with the biggest area being a square, we want also to avoid symmetries along the $y = x$ axis, this is done by enforcing a strict lexicographic order between circuits with flipped dimensions and with coordinates specular to the above-mentioned symmetry axis. To ease the notation, we introduce here a function ψ that swaps coordinates:

$$\begin{aligned} \psi &: \mathbb{N} \times \mathbb{N} \longrightarrow \mathbb{N} \times \mathbb{N} \\ &(a, b) \longmapsto (b, a) \end{aligned} \tag{24}$$

$$\forall i, j \in C : i < j, D_i = \psi(D_j), P_i = \psi(P_j) \implies \texttt{lex\_less}(P_i, P_j) \tag{25}$$

where `lex_less` is a native predicate of Minizinc used to enforce lexicographic ordering.

Finally, we avoid symmetric solutions given by swapping circuits with the same dimension by imposing a lexicographic order among them. In symbols:

$$\forall i, j \in C : i < j, D_i = D_j \implies \texttt{lex\_less}(P_i, P_j) \tag{26}$$

2.4 Rotation

To take into account the eventuality of rotating pieces to achieve a solution, we introduce a vector r of Boolean variables:

$$r \in \mathbb{B}^{|C|} \tag{27}$$

Its function is to indicate whether a circuit c is rotated or not. Then, we define two vectors of variable values to store the new dimensions in the case of rotation:

$$\hat{W} = (\hat{w}_1, ..., \hat{w}_{|C|}) \in \mathbb{N}^{|C|} \qquad \hat{H} = (\hat{h}_1, ..., \hat{h}_{|C|}) \in \mathbb{N}^{|C|} \tag{28}$$

In the code, we call these two vectors `rotation_widths` and `rotation_heights` respectively.
For efficiency, each $\hat{w}_c$, $\hat{h}_c$ spans from the smallest to the largest of the values of height or width among the circuits. In symbols, defining as $\hat{D}_c$ the dimensions of a circuit c in the context of the rotated model, we have:

$$\hat{D}_c = (\hat{w}_c, \hat{h}_c), \quad \hat{w}_c, \hat{h}_c \in [d_{min}, d_{max}] \tag{29}$$

where

$$d_{min} = \min_{c \in C}(w_c, h_c), \quad w \in W, h \in H \tag{30}$$

$$d_{max} = \max_{c \in C}(w_c, h_c), \quad w \in W, h \in H \tag{31}$$

we write the assignment of the new rotated dimensions as $\forall c \in C$:

$$r_c \implies (\hat{w}_c = h_c) \wedge (\hat{h}_c = w_c) \tag{32}$$

$$\neg r_c \implies (\hat{w}_c = w_c) \wedge (\hat{h}_c = h_c) \tag{33}$$

In code, the values to such vectors are assigned using `if...else` Minizinc operators.

Once the rotational case has been modeled, the same constraints of Section 2.3 are applied, using $\hat{w}_c$, $\hat{h}_c$ and the respective ordered lists of circuits widths and heights $\hat{W}$, $\hat{H}$ instead of the nonrotated versions, namely w, h, W, H.

An additional symmetry-breaking constraint is used to avoid the rotation of squared circuits. In symbols:

$$\forall c \in C \colon w_c = h_c \implies \neg r_c \tag{34}$$

2.5 Validation

Experimental design

Experiments were run on a Windows operative system, with a 64-bit architecture, an Intel Core i5 8th gen, and 16GB of RAM. For all the experiments a time limit of 3 minutes. For the experiments, we define several search strategies. In all cases we use `solve_seq`, searching first for values of η using `indomain_min` to select its smallest value. Then we proceed the search for x and y. For both we use the same search strategies.

First, we use `Gecode`, selecting the variable to search by `first_fail`, selecting random in the domain by using `indomain_random`. We use `luby` with a seed of 10 to restart the search (A).

Then, we change our heuristic, switching to `input_order`, by using as selecting order the indexes of the circuits from the biggest to smallest in terms of area. We still use `Gecode` and `luby` with a seed of 10 to restart the search (B).

Next, we use `Chuffed` on the same input order using `indomain_min` to select the values in the respective domains (C).

Moreover, we change our input order using the indexes of the circuits from the biggest to smallest in terms of largest among width and height (D).

And finally, we run a test using `Chuffed` on using descending bigger areas as input orders, by indomain_min and without the symmetry-breaking constraints to test performances (E).

For the rotated model, we use `Chuffed` using the indexes of the biggest areas in descending order and selecting the vales by`indomain_min` (R).

Experimental results

Experimental results are shown in Table 5, where the values for η for each optima configuration are found. From the data, we can see that every search strategy produces an optimal solution when it does produce a solution. Gecode shows the worst performances, with the less number of optimal solutions found in both search strategies (A, B). Chuffed performs well in both search strategies, showing no appreciable differences in terms of the number of optimal solutions (C,D). Not using symmetry-breaking constraints leads to a slight reduction in the number of optima solutions found (E). The rotated model performs slightly worse than the base version (R).

3 SAT Model

Boolean Satisfiability (SAT) involves formulating a problem as a boolean expression and finding an assignment that satisfies it. In code, W, H are stored in two arrays of dimension $|C|$, p and $|C|$ as integer values.

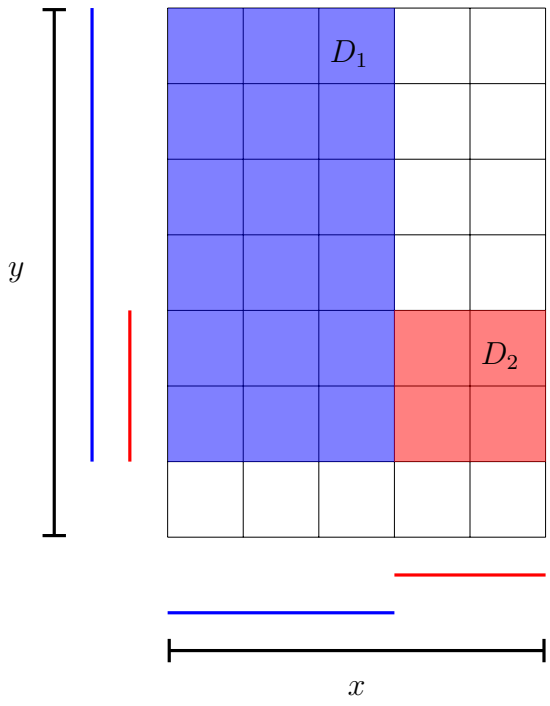

Figure 1: *Two circuits D_1, D_2 placed on a board, along with their respective projection on the x and y axes*

3.1 Decision Variables

The decision variables in our model are the projections of each circuit on the x and y axes. The position of each circuit c is thus encoded in two two vectors:

$$x_c = (x_c^1, .., x_c^p) \in \mathbb{B}^p \qquad y_c = (y_c^1, .., y_c^\eta) \in \mathbb{B}^\eta \tag{35}$$

These vectors represent the portion of an axis filled by the shadow of a circuit once it has been placed. Indexes of portions occupied by the shadow of a circuit will have $True$ values, whereas those not filled, will have $False$ values. In the code, such variables are stored into two matrices for simplicity:

$$\begin{aligned} X \in \mathbb{B}^{p \times |C|}, X = (x_1, ..., x_{|C|}) \\ Y \in \mathbb{B}^{\eta \times |C|}, Y = (y_1, ..., y_{|C|}) \end{aligned} \tag{36}$$

A visual intuition of our decision variables is provided in Figure 1, 2, where we show two circuits $D_1 = (3, 6), D_2 = (2, 2)$ placed arbitrarily on a 5×7

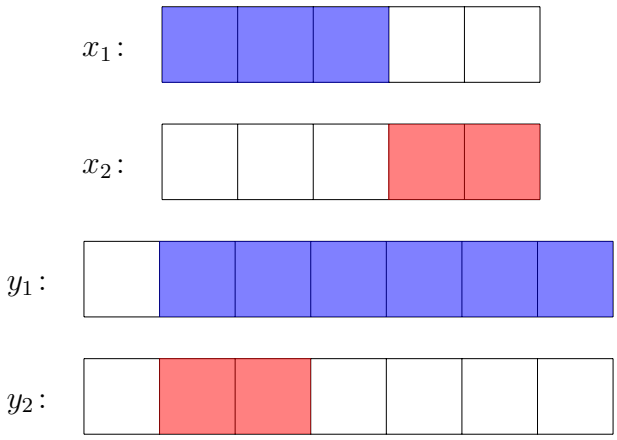

Figure 2: *Graphical representation of the distribution of* $True$ *values in the* x, y *vectors of* D_1, D_2*. Colored cells represent* $True$ *values, white cells represent* $False$ *ones.*

board. In Figure 1 we can see their projections along the axes, whereas in Figure 2 we can appreciate graphically their corespondent x,y decision variable vectors, as defined in (35), for such a disposition. There, colored cells represent $True$ values, and white cells represent $False$ ones.
After imposing the constraints that model the problem, once a solution is found we extract the coordinates of the bottom left corner of a circuit c, namely P_c, as the index of the first $True$ value of x_c and y_c respectively.

3.2 Objective function

As it was for CP, also in this model we chose the height occupied by the circuits once all placed on the board as our objective function to minimize, namely η defined as in (18). As the model relies on $Z3$ solver, η is minimized by simply looking for an assignment that satisfies the problem starting from the lowest value of its codomain. This value is then increased by one if a solution is not found, iterating until the upper bound. Values that depend on η are thus initialized accordingly at each iteration.

3.3 Constraints

The base model works under just two constraints, named `existence` and `impenetrability`. In order to further constraint the search, two implied

constraints were added, namely `strong_existence` and `unicity`. For the mathematical representation of these constraints we use d_c as a generic dimension of a circuit, $v_c \in \mathbb{B}^d$ as a generic variable and ℓ as a generic dimension of the board. These generic values will represent specific entities depending on which of the two matrices X, Y the constraint is enforced. For instance, if `existence` is enforced on X, then d_c, v_c, ℓ will represent w_c, x_c, p respectively.

`existence`

This constraint ensures that for each $c \in C$, v_c will have at least d_c consecutive $True$ values. To ease the mathematical description, lets define the set S^ℓ as the set of all possible coordinates occupied by the projection of a circuit on an axis of length ℓ, namely:

$$S^\ell = \{1, ..., \ell - 1\}, \tag{37}$$

and T_c^i as the set of consecutive indexes representing the shadow of a circuit c on an axis, i.e. the indexes of i d_c consecutive $True$ values:

$$T_c^i = \{t \in S^\ell \colon t \in [i, i + d_c - 1], v_c^t = True\} \tag{38}$$

We finally define the set of all possible positions that the first $True$ value of our d_c-plette can assume in our decision variable vector v_c:

$$I_c^\ell = \{1, ..., \ell - d_c\} \tag{39}$$

An index $i \in I$ defines the starting point of our d_c-plette, with its upper bound restricted to $\ell - d_c$ to reduce the search space. An index j spans along the number of cells to be true according to d_c. In symbols:

$$\bigwedge_{c \in C} \bigvee_{i \in I_c^\ell} \bigwedge_{t \in T_c^i} v_c^t \tag{40}$$

In order to get a better grasp of this constraint and of all its indexes, let's use a toy example. Given $D_1 = (3, 6)$ placed again on a 5×7 board, let's represent a possible solution to the `existence` constraint applied to its x-projection. In Figure 3 we see three different configurations of consecutive $True$ values, represented by colored cells. Below each vector, we can appreciate the progression of the i index, whereas above we can see the values

assumed by the j index as they are defined in (51). Blue values represent indeed the possible horizontal positions of D_1. The last configuration is also a possible solution, besides it contains a $True$ value in a position without a physical interpretation, i.e. the green cell. This is because `existence` requires at least one configuration of consecutive $True$, but does not impose any constraint on the other nonconsecutive cell.

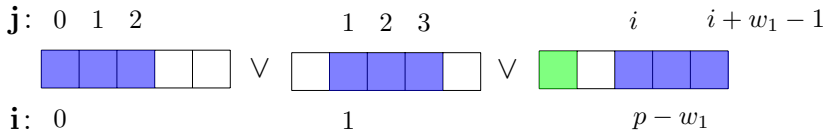

Figure 3: *Graphical representation of an assignment satisfying* `existence` *on* x_1*, given* $D_1 = (3, 6)$*. Cells with* $True$ *values are colored. Below each vector is shown the progression of the index* $i \in I_c^\ell = \{1, 2, 3\}$*. Above, we show the progression of the index* $j \in T_1^i$*.*

`strong_existence`

This constraint ensures that for each $c \in C$, its v_c will have at least d_c consecutive $True$ values, and constrains all the others to be $False$, avoiding an occurrence such that of the green cell in Figure 3. To ease the notation we define F_c^i as the set of indexes in S^ℓ but not in T_c^i:

$$F_c^i = \{f \in S^\ell \colon f \notin T\} \tag{41}$$

Then, using (38), we can write our constraint as:

$$\bigwedge_{c \in C} \bigvee_{i \in I_c^\ell} \bigwedge_{t \in T_c^i} v_c^t \bigwedge_{f \in F_c^i} \neg v_c^f \tag{42}$$

As for before we provide a graphical representation for a assignment that satisfies `strong_existence` for $D_1 = (3, 6)$ placed on a 5×7 board. In Figure 4 we can see in blue, values restricted to be $True$, whereas in pink, those constraint to be $False$.

Figure 4: *Graphical representation of an assignment satisfying* `strong_existence` *on* x_1*, given* $D_1 = (3, 6)$*. Cells with* $True$ *values are blue, and those restricted to False are pink.*

`unicity`

This constraint ensures that at most one configuration of d_c consecutive $True$ values exists for each $c \in C$, using pairwise encoding. To ease the notation, let's define a d_c-plette of i dependant consecutive $True$ values as:

$$\tau_c^i = \bigwedge_{t \in T_c^i} v_c^t \tag{43}$$

where $i \in I_c^\ell$ one of the possible positions of the first $True$ value for a circuit with dimension d_c in the projection on the side of the board of length ℓ. For instance, τ_1^1, would represent the first 3-plette of $True$ values in Figure 4, τ_1^2 would represent the second, and τ_1^3 the third. Then we define the set of all possible combinations of the index i, to be used to ensure that only one satisfies `unicity` for each circuit. Given (39):

$$\mathfrak{I}_\mathfrak{c} = \binom{I_c}{2} \tag{44}$$

Thus, our `unicity` constraint can be expressed as:

$$\bigwedge_{c \in C} \bigwedge_{i,j \in \mathfrak{I}} \neg(\tau_c^i \wedge \tau_c^j) \tag{45}$$

`impenetrability`

Finally, our more characterizing constraint, which ensures that for every combination of circuits i, j, if they share at least one index k such that $x_i^k = x_j^k = True$ then it can not exist an index $z : y_i^z = y_j^z = True$. Also, in this case, we used pairwise encoding. To ease the notation, we define the set of all the possible 2-combinations of circuits:

$$\mathfrak{C} = \binom{C}{2} \tag{46}$$

$$\bigwedge_{c_1,c_2\in\mathfrak{C}}\left[\bigvee_{k=1}^{p}(x_{c_1}^k\wedge x_{c_2}^k)\implies\bigwedge_{z=1}^{\eta}\neg(y_{c_1}^z\wedge y_{c_1}^z)\right] \tag{47}$$

We can read this constraint as follows: taking two circuits c_1, c_2, we see if their x-projection, x_{c_1}, x_{c_2}, overlap somewhere, they do, then their y-projections y_{c_1}, y_{c_2} must not overlap in anywhere.

In Figure 5 we can see a case where circuits do overlap, and indeed they do not satisfy our `impenetrability` constraint. For instance, the red and yellow x-projections do overlap, but their y-projection counterpart do not have empty intersection, indeed the red and the yellow circuit do also overlap on the board.
In Figure 6 we can see how for a disposition of nonoverlapping circuits, our `impenetrability` constraint does hold. Indeed for every pair of x-projections that do overlap, their y-projections counterparts do not overlap. This works also for suboptimal configurations.

Thus, by using `existence`, by restricting the domain of the possible projections and by enforcing `impenetrability` we are finally able to model and solve VLSI in a SAT fashion.

3.4 Rotation

The case that allows the rotation is modeled through the use of a vector of boolean variables, that indicates if a circuit is rotated or not.

$$r_c = (r_1, ..., r_{|C|}) \in \mathbb{B}^{|C|} \tag{48}$$

Then we slightly modify `existence`. To ease the notation, given d_c, which is a generic dimension of a circuit c, we define $\bar{d}_c$ as the other dimension. Thus, if $d_c \equiv w_c$, then $\bar{d}_c \equiv h_c$ and viceversa. Also, given T_c^i, I_c^ℓ as they were define in (38, 39), we also define their dual versions as:

$$\bar{T}_c^i = \{t \in S^\ell \colon t \in [i, i + \bar{d}_c - 1], v_c^t = True\} \tag{49}$$

$$\bar{I}_c^\ell = \{1, ..., \ell - \bar{d}_c\} \tag{50}$$

Thus, our `existence` constraint can be redefined as:

$$\bigwedge_{c\in C}\left[\left(\bigvee_{i\in I_c^\ell}\bigwedge_{t\in T_c^i}(v_c^t\wedge\neg r_c)\right)\oplus\left(\bigvee_{i\in \bar{I}_c^\ell}\bigwedge_{t\in \bar{T}_c^i}(v_c^t\wedge r_c)\right)\right] \tag{51}$$

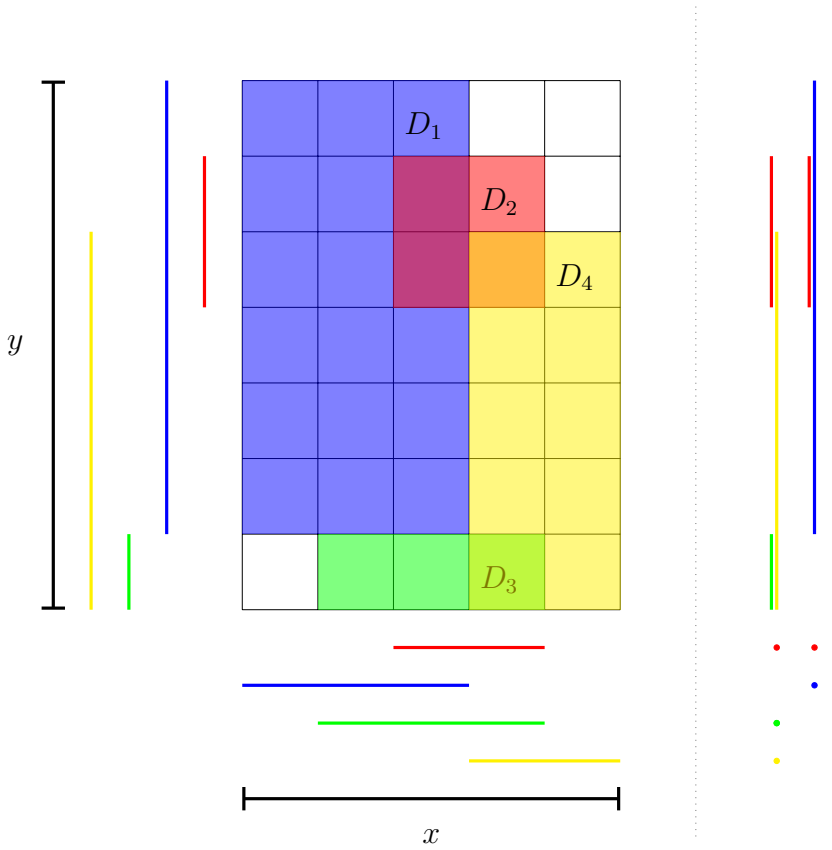

Figure 5: *Circuits $D_i, i \in 1, ..., 4$ on a board, along with their respective projection on the x and y axes. The configuration is sub-optimal and the circuits are overlapping, so impenetrability is not satisfied. This can be seen, for instance, by checking that overlapping x projections, like red and yellow, have a nonempty intersection between their respective y projections.*

What it enforces is that, for each circuit, if it is rotated, we look for its existence only by using the correspondent rotated dimension. The *xor* ensures that only one case is possible for each circuit.

For rotation we just implement a base version using just `existence` and `impenetrability`, which remains the same.

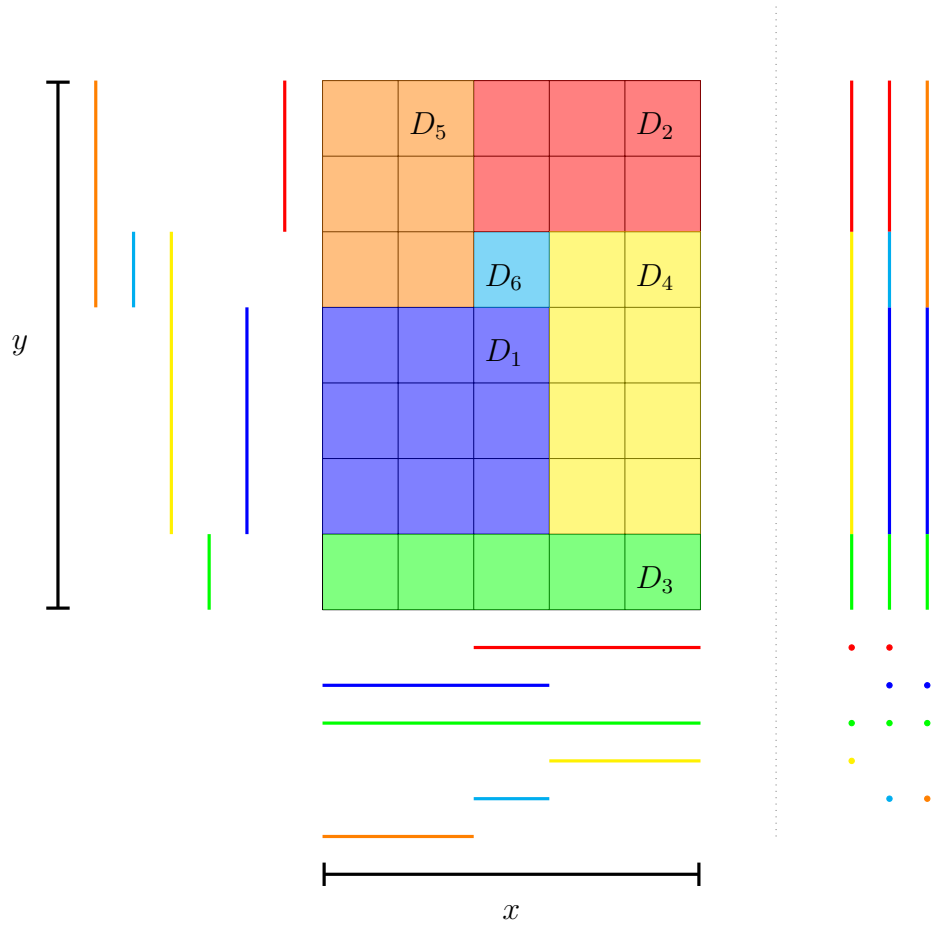

Figure 6: *Circuits $D_i, i \in \{1, ..., 6\}$ placed on a board in optimal configuration, along with their respective projection on the x and y axes. Each pair of distinct circuits whose x projections are overlapping have an empty intersection between their respective y projections.*

3.5 Validation

The model is implemented by using z3. Two different versions were tested for the non-rotated model and one for the rotated. In the first test, we use just `existence` on the matrix X and on the matrix, Y defined in (36) and `impenetrability`, we refer to this as "base". In the second we use `strong_existence` and `unicity` on X, Y and `impenetrability`, we refer to this as "strong". Finally, for the rotated model, we use just `existence` and

`impenetrability`, namely "rotated". Results are shown in Table 2. We can see that all the models provide an optimal solution when they do provide on in the timelimit. Adding more constraints such as `strong_existence` and `unicity` doesn't improve the model. The rotated version performs slightly worse than the base one.

4 MIP Model

In this section, we address the task with a Mixe-Integer Approach. As usual, we created two models: one standard and one to account for the rotation of the circuits. Both models are based on PuLP, a Python linear programming toolkit that interfaces with different open sources and proprietary solvers.

4.1 Decision variables

We defined 3 different decision variables: 1 integer variable for the height, with a lower and upper bound, and 2 sets of integer variables that encodes the coordinates of the bottom left point of each circuit in the problem.
To model our problem, we chose as decision variables the coordinates of the bottom left corner of each circuit, as it was for CP. In code, these variables are encoded in two vectors of variable values x, y defined as in (10). The domains of values that each coordinate can take are limited as it was in (12, 13), but this time we tailor it for each circuit:

$$x_c \in [0, p - w_c] \subset \mathbb{Z} \tag{52}$$

$$y_c \in [0, \eta - h_c] \subset \mathbb{Z} \tag{53}$$

4.2 Objective function

As it was for CP, we define our objective function as the height occupied by the circuits once they are all packed on the board according to all the constraints. We call this function η and we define it as it was in (18). Again, the ultimate goal of our solver is to minimize η, i.e. to find an assignment of values for P, defined as in (11) such that, given D as defined in (6), the value returned by $\eta(P, D)$ is minimum.

4.3 Constraints

`bounds`

Bounds ensure that each circuit is placed inside the board, once a value for η is given. It does so by defining a boolean vector and constraining its value to $True$ whenever a circuit c is inside the board. In symbols:

$$b = (b_1, ..., b_{|C|}) \in \mathbb{B}^{|C|} \tag{54}$$

$$\forall c \in C \qquad b_c \iff y_c \leq \eta - h_c \tag{55}$$

`no_overlap`

To ensure that circuits do not overlap, we have to ensure that for any two circuits $c_1, c_2 \in \mathfrak{C}$ defined in (46), the following holds:

$$\begin{aligned} &(x_{c_1} + w_{c_1} \leq x_{c_2}) \\ \vee &(x_{c_2} + w_{c_2} \leq x_{c_1}) \\ \vee &(y_{c_1} + h_{c_1} \leq y_{c_2}) \\ \vee &(y_{c_2} + h_{c_2} \leq y_{c_1}) \end{aligned} \tag{56}$$

In code we enforce these constraints using the Big-M method. Such a method is a mathematical optimization technique used to model constraints in the case of variables having an upper or lower bound. To use the big M method we define a binary variable:

$$\delta_{c_1,c_2,k} \in \{0, 1\} \tag{57}$$

where k is an index that runs for each possible orientation up, down, left, and right. This binary variable takes a value of 1 if the circuits overlap and a value of 0 if they do not. The goal of our solver will then be to minimize this binary variable, subject to constraints that enforce the non-overlap of the two circuits. The value of M in the big M method refers to a large positive constant used in the constraints to enforce the binary nature of the variable. Thus, we enforce the following constraints

$$\begin{aligned} x_{c_1} + w_{c_1} &\leq x_{c_2} + M_1\delta_{c_1,c_2,1} \\ x_{c_2} + w_{c_2} &\leq x_{c_1} + M_2\delta_{c_1,c_2,2} \\ y_{c_1} + h_{c_1} &\leq y_{c_2} + M_3\delta_{c_1,c_2,3} \\ y_{c_2} + h_{c_2} &\leq y_{c_1} + M_4\delta_{c_1,c_2,4} \end{aligned} \tag{58}$$

$$\sum_{k}^{4} \delta_{i,j,k} \leq 3 \tag{59}$$

4.4 Rotation

As mentioned before, we developed also a model that allows the rotation of the circuits. Its structure follows the same logic as the standard model, but of course, we needed to make little modifications to the definition of the decision variables and all of the constraints. To allow it to work we define a variable vector of bools:

$$r = (r_1, ..., r_{|C|}) \in \mathbb{B}^{|C|} \tag{60}$$

The idea is to use it to govern the behavior of the no-overlap function and bound constraint on whether to use the width or the length of a circuit when trying to enforce (58). So for each of the lines in (58) instead of just ensuring that each dimension is inside its bounds, guided by M, we add the dual dimension depending on the behaviour of r_c.

We also modify "bounds" to take into account different possible orientations for our circuits. Using

$$\forall c \in C \qquad \mathfrak{b} \iff v_c - d_c r_c + \bar{d}_c r_c \leq d_c + \ell \tag{61}$$

where

$$\mathfrak{b} \in \mathfrak{B}^{|C|} \tag{62}$$

is a generic boolean vector indicating if a circuit is bounded, v_c is a generic coordinate, d_c a generic dimension and d_c its dual version as used in SAT constraints description, ℓ a generic dimension of the board. Due to time constraints, we were not able to perform tests on such model.

4.5 Validation

We tested our models with Gurobi (with academic license), the state-of-the-art solver for this kind of problem, and PULP_CBC_CMD. Results are summarized in Table 3.

5 Conclusions

CP and MIP were very similar. MIP offers more flexibility, but with a cost in language complexity, whereas CP is more tailored to optimization problems, indeed it has better results and is more intuitive. SAT was the more fun to work with, as the modeling phase involves a good dose of creativity. Our best results are listed in Table 4. In Figure 7 we can appreciate some pictures of the solutions found by our models.

ID	B	C	E	A	D	R
1	8	8	8	8	8	8
2	9	9	9	9	9	9
3	10	10	10	10	10	10
4	11	11	11	11	11	11
5	12	12	12	12	12	12
6	13	13	13	13	13	13
7	14	14	14	14	14	14
8	15	15	15	15	15	15
9	16	16	16	16	16	16
10	17	17	17	17	17	17
11	18	18	18	18	18	18
12	19	19	19	19	19	19
13	20	20	20	20	20	20
14	21	21	21	21	21	21
15	22	22	22	22	22	22
16	23	23	23	23	23	23
17	24	24	24	24	24	24
18	25	25	25	25	25	25
19	N/A	26	26	26	26	26
20	27	27	27	27	27	27
21	28	28	28	28	28	28
22	N/A	29	29	N/A	29	N/A
23	30	30	30	30	30	N/A
24	31	31	31	31	31	31
25	N/A	32	32	32	32	N/A
26	33	33	33	N/A	33	33
27	34	34	34	34	34	34
28	35	35	35	N/A	35	35
29	36	36	36	N/A	36	36
30	N/A	37	37	N/A	37	N/A
31	38	38	38	38	38	38
32	N/A	39	39	N/A	39	39
33	40	40	40	40	40	40
34	40	40	40	40	40	40
35	40	40	40	40	40	40
36	40	40	40	40	40	40
37	N/A	60	60	N/A	60	N/A
38	N/A	60	N/A	N/A	60	N/A
39	N/A	60	60	N/A	60	N/A
40	N/A	N/A	N/A	N/A	N/A	N/A

Table 1: Optimal solution for η for different solving searches as defined in Section 2.5.

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

Table 2: List of optimal solution for η for three tests of SAT model.

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

Table 3: Results for different solver of MIP.

ID	D	Base	Gurobi
1	8	8	8
2	9	9	9
3	10	10	10
4	11	11	11
5	12	12	12
6	13	13	13
7	14	14	14
8	15	15	15
9	16	16	16
10	17	17	17
11	18	18	18
12	19	19	19
13	20	20	20
14	21	21	21
15	22	22	22
16	23	23	23
17	24	24	24
18	25	25	25
19	26	26	26
20	27	27	27
21	28	28	28
22	29	29	N/A
23	30	30	N/A
24	31	31	N/A
25	32	32	N/A
26	33	33	N/A
27	34	34	N/A
28	35	35	N/A
29	36	36	N/A
30	37	N/A	N/A
31	38	38	N/A
32	39	N/A	N/A
33	40	40	N/A
34	40	40	N/A
35	40	40	N/A
36	40	40	N/A
37	60	N/A	N/A
38	60	N/A	N/A
39	60	N/A	N/A
40	N/A	N/A	N/A

Table 4: Summarizing best results for each approach.

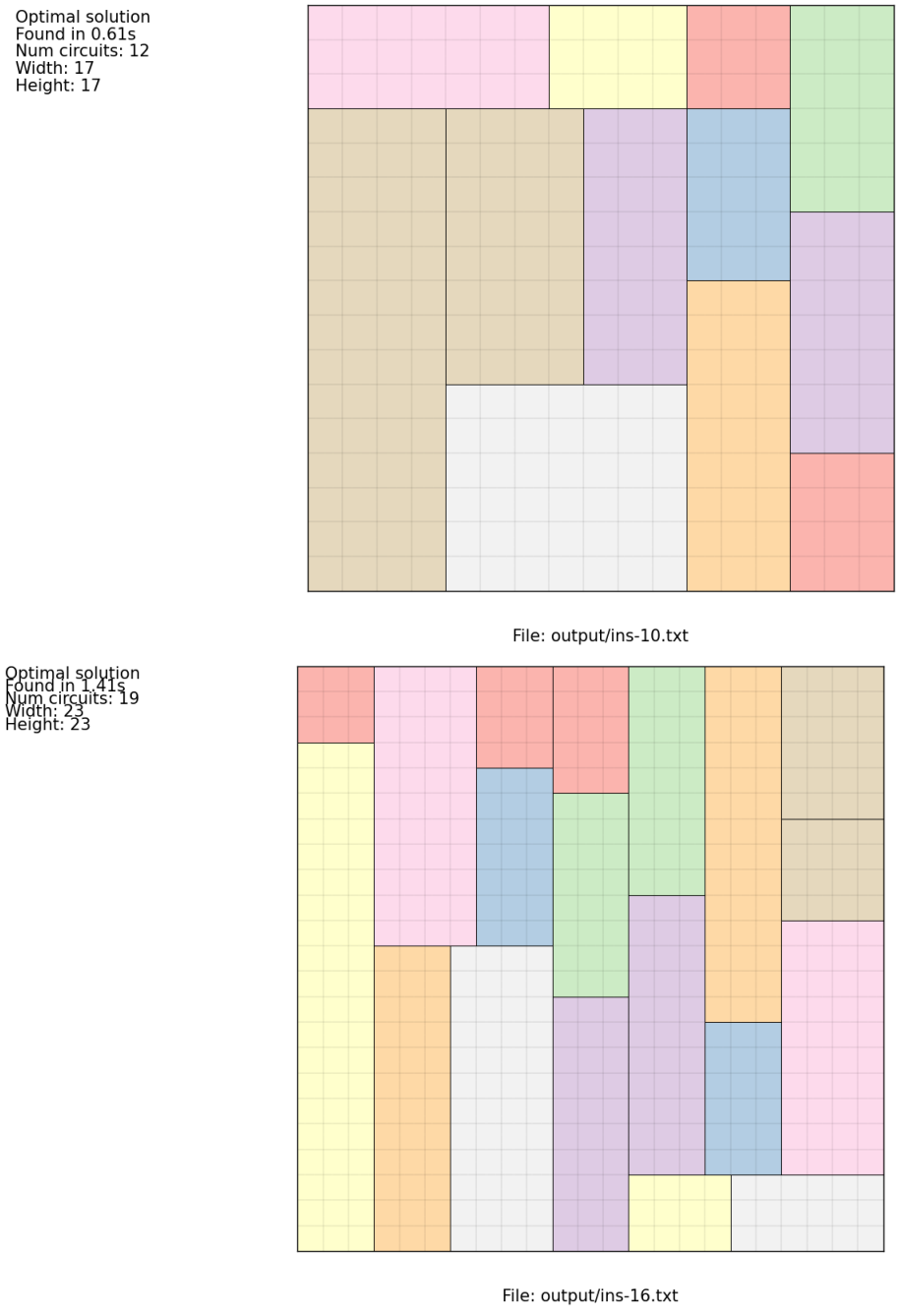

Figure 7: Visualization of an optimal solution on 10^h and 16^{th} input.