Combinatorial Decision Making and Optimization

Ole August Støle, Andrea Stette Jessen, Solveig J. Mohr

May 2022

Contents

Problem description

This project aims to solve Very Large Scale Integration (VLSI) design problems. The problem resolves around placing a defined number of rectangular circuits of different sizes on a plate with a fixed width. The goal is to minimize the length of the plate. Two variants of the problem have been considered, one allowing rotation of the circuits and the other with a fixed orientation for each circuit. The VLSI design problem have been solved using Constraint Programming (CP), propositional SATisfiability (SAT), and Mixed-Integer Linear Programming (MIP).

Problem parameters The problem parameters for the VLSI design problem are equal for the CP, SAT and MIP implementations. The problem parameters are the width of the plate (w), the number of circuits to be placed (n) and the dimensions (width and height) of each circuit.

Decision variables The decision variables will concern placement of each circuit. How these are defined will vary for the different implementations.

Main problem constraints In a valid solution no circuits overlap, no circuit exceeds the width of the plate, the circuits are coherent and can not be split into smaller parts, and all n circuits must be placed on the plate. How these requirements are translated into constraints differ for CP, SAT and MIP.

Objective Variable The objective variable, l, for all implementations is the the total height of the fixed-width plate.

Objective Function The objective function is to minimize the total height of the plate (l).

Upper and lower bound The lower bound of the objective function is the same for all the implementations. First, the minimum number of rows needed on the plate is found. This is calculated with the formula

$$n\_rows = \frac{\sum_{c=1}^{n} width_c}{w}, \tag{1}$$

where $width_c$ is the width of a circuit and w is the fixed width of the plate. After acquiring the minimum number of rows needed, we sum the n_rows lowest heights,

$$sorted\_heights =< h_0, ..., h_i, h_j ..., h_n >, h_i \leq h_j, i \neq j \tag{2}$$

$$lower\_bound = \sum_{c=1}^{n\_rows} sorted\_heights_c. \tag{3}$$

This results in the optimal lower bound for any instance.

The upper bound is simply the sum of all the circuits' height. This is the worst case scenario.

$$\sum_{c=1}^{n} height_c \tag{4}$$

CP

Constraint programming (CP) was the first approach used to solve the VLSI design problem. This solution was developed in Minizinc. The differences in the implementations of the fixed-orientation problem and the problem allowing rotation will be highlighted. There are several models for each problem, where each model is a result of an attempt to improve the first. The solver that performed best on the first model was Chuffed, therefore this solver is used throughout the experiment. As mentioned in the problem description the problem parameters are the width of the plate (w), the number of circuits (n) and the dimensions of the circuits ($dims$). The CP problem is formalized as $< X, D, C, f >$, where X is a set of the decision variables, D is the domains for the decision variables, C is the constraints and f is the objective function. The *lower_bound* and *upper_bound* is the same as described in the problem description, respectively equation 3 and 4. In this description of the CP solution, w_i and h_i will be used for representing the width and height of circuit i, respectively. w_i and h_i depends on whether or not rotation is allowed and are defined as shown below.

$$w_i = \begin{cases} dims_{i,1} & \text{if } rotation\_allowed = false \\ dims_{i,rotation_i+1} & \text{if } rotation\_allowed = true \end{cases}$$

$$h_i = \begin{cases} dims_{i,2} & \text{if } rotation\_allowed = false \\ dims_{i,2-rotation_i} & \text{if } rotation\_allowed = true \end{cases}$$

Decision Variables

In the fixed-orientation problem the decision variables are $X = \{pos\_x, pos\_y\}$. If rotation is allowed one additional decision variable is introduced, making $X_r = \{pos\_x, pos\_y, rotation\}$. Pos_x, pos_y and $rotation$ are all arrays of length n. The i'th element of pos_x and pos_y are the x_i and y_i position of the lower left corner of circuit i. $Rotation_i$ is equal to 1 if circuit i is rotated and 0 if not. The domains of the variables for circuit i are:

$$D(x_i) = [0 \dots w]$$
$$D(y_i) = [0 \dots upper\_bound]$$

$$D(rotation_i) = \{0, 1\}$$

Objective Variable and Function

The objective variable l is an integer representing the height of the fixed-width plate. The domain of l is $D(l) = [lower\_bound \ldots upper\_bound]$. The objective variable is defined as follows,

$$l = max(\text{for all } i \in \{1, \ldots, n\}, y_i + h_i)$$

The objective function is $f : minimize\ l$.

Problem Constraints

There are two main constraints for this problem. First of all it is necessary to ensure that no circuit violates the max width of the plate. This yielded in constraint C_1, which restricts every x-position added with the width of the circuit to not be greater than the width of the plate, w.

$$C_1 : \text{for all } i \in \{1, \ldots, n\}, x_i + w_i \leq w$$

Secondly, one needs to make sure there is no overlap between the circuits. Constraint C_2 is for the fixed-orientation problem, whereas constraint C_{2r} is for the rotation problem. The overlap constraint is done differently in regard to whether or not rotation is allowed. When rotation is allowed it is necessary to check if the circuits overlap both when rotated and not, as the circuit may fit if it is rotated. In addition, *rotation* needs to be updated so that a circuit is rotated in regard to every other circuit.

$$C_2 : \text{for all } i < j \in \{1, \ldots, n\}, \text{not } overlapping(i, j)$$

$$C_{2r} : \text{for all } i < j \in \{1, \ldots, n\}, (\text{not } overlapping(i, j) \wedge rotation_j = 0)$$

$$\vee(\text{not } overlapping(i, j) \wedge rotation_j = 1)$$

Overlapping is a predicate which returns whether or not circuit i and circuit j is overlapping. This is done by checking if the circuits overlap in x-direction and y-direction at the same time.

$$x\text{–}direction : x_i + w_i > x_j \wedge x_j + w_j > x_i$$
$$y\text{–}direction : y_i + h_i > y_j \wedge y_j + h_j > y_i$$

Constraint C_1 and C_2/C_{2r} make out the basis model used. Further, different adjustments are done in an attempt to improve this model.

Implied constraints First of all the following implied constraint, C_3 came to attention,

$$C_3 : alldifferent(pos)$$

Pos is an array of (x_i, y_i) representing the position of the lower left corner for circuit i. Constraint C_3 make sure no circuit have the same lower left corner.

Global constraints In order to utilize global constraints as much as possible, the constraint guaranteeing no overlap, C_2, was modified. This resulted in C_4 and C_{4r} for the fixed-orientation problem and rotation problem respectively. These constraints use the global constraint *diffn*. It is necessary with a small modification to the constraint when rotation is allowed because some of the circuits might be rotated and some might not be. Therefore one need to retrieve the width and height for each circuit separately in order to get the right dimension.

$$C_4 : diffn(pos\_x, pos\_y, w, h)$$
$$C_{4r} : diffn(pos\_x, pos\_y, w_i\ i \in \{1, \ldots, n\}, h_i\ i \in \{1, \ldots, n\})$$

Symmetry Finally, symmetry breaking was applied in order to make the search space smaller. The first symmetry handled was vertical and horizontal flip of the plate. This was done by lexicographical sorting of the x positions and the y positions. This constraint, C_5 can be formalized as follows,

$$C_5 : lex \leq (pos\_x, x_i\ i \in \{n, \ldots, 1\}) \wedge lex \leq (pos\_y, y_i\ i \in \{n, \ldots, 1\})$$

The second symmetry handled is rotation of squares. This is formalized in constraint C_{6r}.

$$C_{6r} : \text{for all } i \in \{1, \ldots, n\}, w_i = h_i \rightarrow rotation_i = 0$$

Results

There are in total eight different models created, four models for the fixed-orientation problem and the same four model variations for the rotation problem. The 40 instances provided have been tested on all models, where all models are given a time limit of 5 minutes to solve each instance. The four different evaluated model variations are the basis model, the basis model with the implied constraint added (C_3), the basis model with global constraints (C_4, C_{4r}) and lastly, the basis model with global constraints and symmetry breaking (C_5, C_{6r}). Table 1 and table 2 show the number of instances solved within the time limit for each model for the fixed-orientation problem and the rotation problem respectively.

	Basis Model	Implied	Global	Global & Symmetry
Number of instances solved	25	25	24	22

Table 1: Number of instances solved by the different models for the fixed-orientation problem.

	Basis Model	Implied	Global	Global & Symmetry
Number of instances solved	10	11	10	10

Table 2: Number of instances solved by the different models for the rotation problem.

Figure 1 shows the time in seconds each model used to solve each instance for the fixed-orientation problem. In order to see the differences between the different models more clearly figure 2 displays the time used per instance for instance 1 to 9. The same type of plots can be found for the rotation problem in figure 3 and figure 4. The instances that timed out stop at 300 seconds.

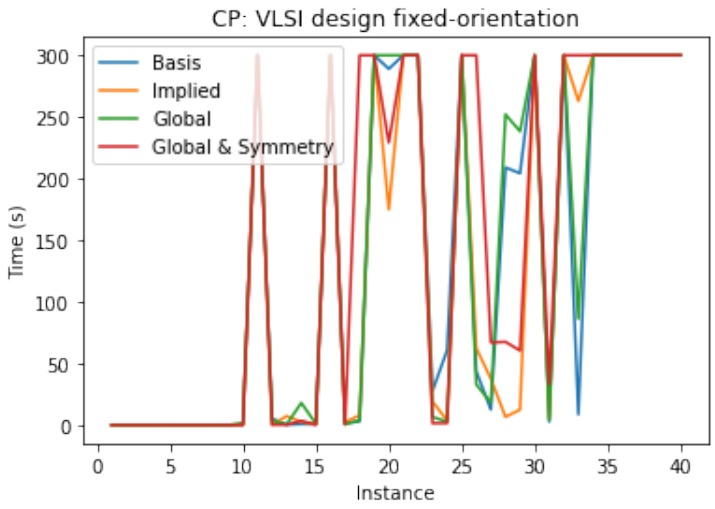

Figure 1: Solving time per instance for the fixed-orientation problem

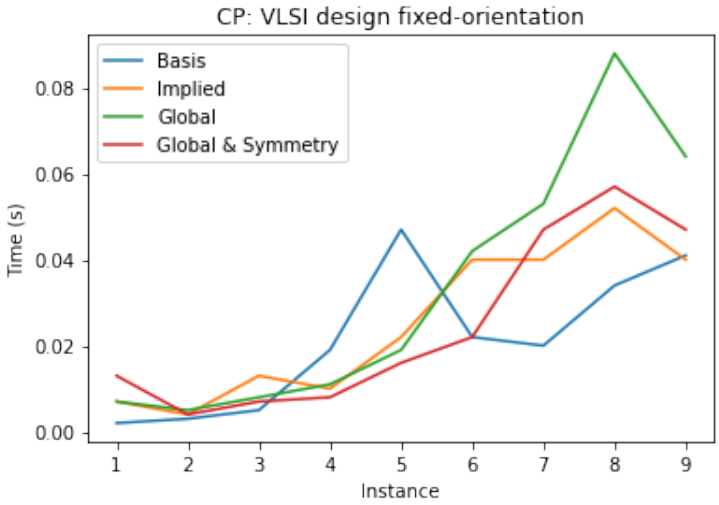

Figure 2: Solving time per instance 1 - 9 for the fixed-orientation problem.

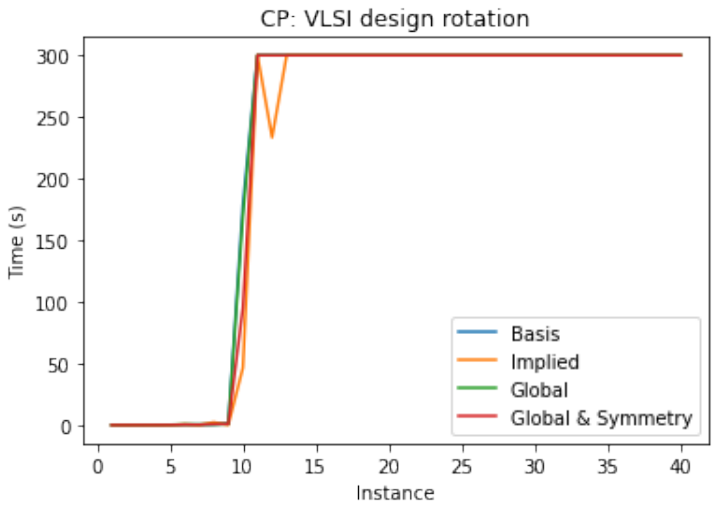

Figure 3: Solving time per instance for the rotation problem.

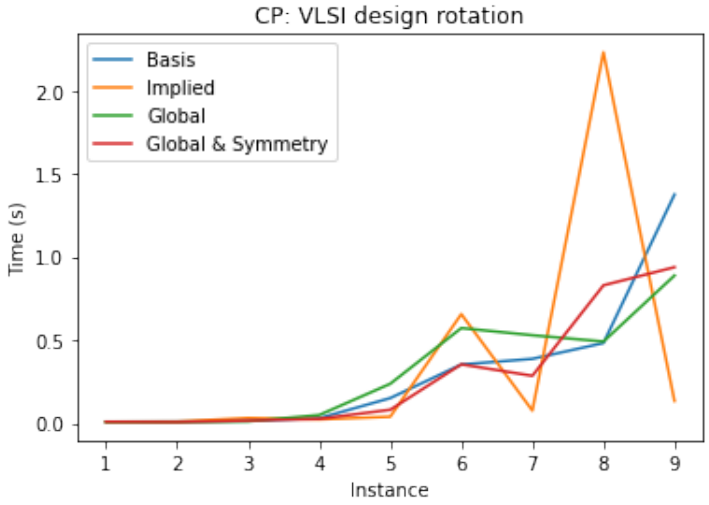

Figure 4: Solving time per instance 1 - 9 for the rotation problem.

Discussion

As one can observe in table 1 and table 2 there are small differences in the number of instances solved within each problem. However the models are

able to solve significantly more instances in the fixed-orientation problem. This is to be expected as the search space without rotation is much smaller and the search time will thereby be smaller as well. This makes it possible for the solver to solve more instances within the time limit of 300 seconds. From figure 1 it can be observed that the solver manage to solve some bigger instances, while it times out on some of the smaller instances. This might be because even though some of the instances have a higher number of rectangles, there may be fewer possible placement combinations, making the search space smaller. However, as can be observed in figure 3, when rotation is allowed the solver is only able to solve the smallest instances. Figure 2 and figure 4 show that there are relatively small time differences between the different implementations of the models. Which results in a conclusion that the adjustment made in an attempt to improve the solving time, did not provide any significant improvements. The differences in solving time increases for the bigger instances for the fixed-orientation problem, figure 1, but it differs between the instances which model is faster. As the differences observed in the plots are small and does not provide any clear tendencies for which model is best, it is likely that the differences may be related to the computer being utilized. In addition the speed of the computations vary depending on which other processes are running on the computer.

SAT

SAT is the second approach used to solve the VLSI design problem. The solution was developed by leveraging Z3 Solver[2].

Implementation

The problem parameters, objective variable and objective function are the same as described in the problem description. The width of the plate (w), the number of circuits to be placed (n) and the dimensions (width and height) of each circuit are stored as local integer variables.

The objective variable l is an integer representing the height of the fixed-width plate. The domain of l is $D(l) = [lower\_bound \ldots upper\_bound]$. The objective variable is initially set to be equal the lower bound, Equation 3.

$$l = lower\_bound$$

In order to model the problem, the plate is represented as a grid of cells, each cell indexed by a x-, y- and c-value. x and y represents the coordinate position in the grid, while c represents the circuit ($c = 0$ representing a grid cell without a placed circuit). In SAT, every decision variable is a boolean literal. As a consequence of this, a list v indexed by $v_{x,y,c}$, is created to contain the decision variables.

$$0 \leq x \leq w, \qquad 0 \leq y \leq upper\_bound, \qquad 0 \leq c \leq n+1$$

The chosen decision variable encoding yields a memory complexity of

$$\mathcal{O}(upper\_bound \cdot w \cdot (n+1)) \tag{5}$$

In order to find the optimal solution, the minimal l satisfying all the constraints, we implemented a search loop. With l initialized as $lower\_bound$ the solver attempts to find a satisfiable solution. If a satisfiable solution is found, that is by definition the optimal solution. If there's not a satisfiable solution for the current l, l is incremented by 1 and a new search is performed. If no optimal solution is found within 5 minutes, the search is timed out.

Problem Constraints

With the selected encoding, three problem constraints need to be ensured.

- C1: There can be at most one placed circuit in each grid cell
- C2: A circuit can't distribute itself onto more grid cells than its given dimensions
- C3: Grid cells containing pieces of circuits need to be coherent and have the correct dimensions.

C1: At most one placed circuit in each grid cell Indexing the decision variable list v on the x and y position yields the list containing n variables, representing the grid cell's contained circuit. By constraining this list to

contain at most one variable to be true, it is ensured that each grid cell doesn't contain more than one circuit.

$$C_1 : \forall_{x,y} AtMostOne(v_{x,y}) \tag{6}$$

Since we allow empty grid cells we don't enforce an *ExactlyOne* constraint.

C2: A circuit can't distribute itself onto more grid cells than its given dimensions The number of grid cells containing a circuit should be equal to the circuit's *area*.

$$C_{2^*} : \forall_{x,y,c} ExactlyK(v_{x,y,c}, area_c),\ area_c = height_c \cdot width_c$$

This encoding ensures the valid constraint, but is not effective as it implicitly constructs a 3-dimensional for-loop, with a time complexity equal to the memory complexity in Equation 5. We can construct a more effective encoding by instead focusing on the grid cells without a placed circuit. Since the plate is a rectangle, and we know the number of circuit grid cells we also know the correct number of empty grid cells.

$$n_{circuit\_cells} = \sum_c area_c \qquad n_{empty\_cells} = (w \cdot l) - n_{circuit\_cells}$$

As a consequence we can reduce the time complexity into $\mathcal{O}(upper\_bound \cdot w)$ by enforcing

$$C_2 : \forall_{x,y} ExactlyK(v_{x,y,0}, n_{empty\_cells}) \tag{7}$$

C3: Grid cells containing pieces of circuits need to be coherent and the correct dimensions To ensure coherent circuits, in accordance to the circuits' dimensions, we compute all their possible placements and select exactly one of them. Given the circuit c, compute the ranges on the plate where the circuit is able to fit.

$$r_x = [0 \leq x \leq w - (width_c - 1)]$$
$$r_y = [0 \leq y \leq upper\_bound - (height_c - 1)]$$

Use the ranges in order to create all the predicates of possible positions within the plate for a circuit c.

$$postions = \forall_c \bigvee_{x \in r_x, y \in r_y} \bigwedge_{i \in width_c, j \in height_c} v_{x+i,y+j,c} \tag{8}$$

Leverage the *ExactlyOne* constraint to select one of the possible positions.

$$C_3 : \forall_c ExactlyOne(positions_c) \tag{9}$$

Rotation The previously discussed constraints are in the case where rotation of a circuit is forbidden. Both C1 and C2 remains the same independent on using rotation or not, but C3 requires a different encoding.

In order to extend C3 to account for rotation, we first check whether the circuit has dimensions which allows it to be rotated on the plate, i.e. that the height of the unrotated circuit isn't greater than the plate's width.

$$r\_pos_c = \forall_{x \in r_x, y \in r_y}((x + height_c) < w) \wedge ((y + width_c) < upper\_bound))$$

If rotation is possible, $r\_pos_c = true$, the rotated circuit predicates are additionally computed, almost as in Equation 8 with the exception that the addition of the circuit's height and width are switched. The rotated predicates and the normal predicates are then concatenated and it's ensured exactly one of the positions are chosen.

$$positions\_rot = \forall_c \bigvee_{x \in r_x, y \in r_y} \bigwedge_{i \in width_c, j \in height_c} v_{x+j,y+i,c}$$

$$C_{3r} : \forall_c ExactlyOne(positions\_rot_c \vee positions_c) \tag{10}$$

Symmetry breaking In order to find all optimal solutions for an instance; when one solution is found, it is appended to a solutions-array before the solution is negated and appended to the current model constraints. The search then continues until the model is unsatisfiable and all the instance's

optimal solutions are stored in the solutions-array. This approach doesn't take into account the symmetry of the optimal solutions.

In order to break symmetry in the solutions, symmetry-breaking constraints are appended. These constraints check for symmetric solutions over the vertical and horizontal axis, when searching for all solutions.

Cardinality constraints

The first implementation of *AtMostOne* was a naive approach, the *Pairwise Encoding*. Since any combination of 2 variables can't be true at the same time, all the combinations of the literals were computed, combined by logical conjunction and negated. The complexity of this operation is $\mathcal{O}(n^2)$[3].

$$\bigwedge_{1 \leq i < j \leq n} \neg(x_i \wedge x_j) \tag{11}$$

By instead implementing the constraint with a *Sequential Encoding*, the complexity of this operation is reduced to $\mathcal{O}(n)$. This approach introduce n new variables s_i to indicate the sum has reached 1 by i[3].

$$\bigwedge_{1 < i < n} x_1 \Rightarrow s_1 \wedge [((x_i \vee s_{i-1}) \Rightarrow s_i) \wedge (s_{i-1} \Rightarrow \neg x_i)] \wedge (s_{n-1} \Rightarrow \neg x_n) \tag{12}$$

Results

All 40 provided instances have been tested using SAT for both VLSI design problem with and without rotation. A time limit of 5 minutes have been introduced for the program to solve an instance. If the program uses more than 5 minutes to solve an instance, it will stop and mark the instance as unsolved. After observing that a lot of the 5 minutes were being spent on searching for the correct l, we additionally experienced with adding 20 seconds timeout per attempted lower bound.

NB: The time used (y-axis) is only for the search of a given l, not the entire instance search.

	No rotation	Rotation	No rotation, timeout_{20s}
Num. instances solved	12	9	13

Table 3: Number of instances solved by the different models for the rotation problem.

Instance number	1	2	3	4	5
With symmetry breaking	4	12	27	61	164
Without symmetry breaking	12	24	73	165	368

Table 4: Number of instances found with/without symmetry breaking constraints, with rotation enabled

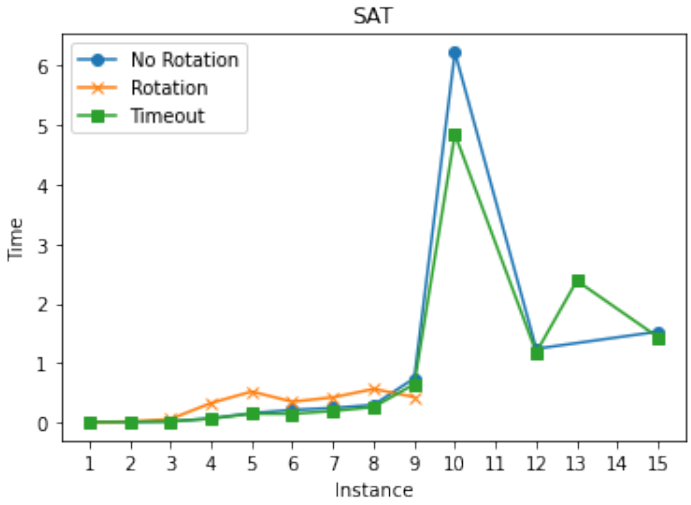

Figure 5: Solving time per instance

Discussion

The solver isn't able to solve more than approximately 25% of the instances. As seen in, Table 3, there are some subtle differences between the encodings that enable rotation. The initial implementation, not accounting for rotation, is able to solve an additional 3 instances. This is likely due to the extra predicates that are included with the introduction of rotation, which creates more predicates to propagate. The symmetry-breaking constraints doesn't

affect the results except for when we search for all solutions. The additional symmetry-breaking constraints lower the total amount of solutions found as seen in Table 4.

MIP

Implementation

The VLSI design problem was also solved using mixed-integer linear programming (MIP). A solution was developed for the VLSI problem with and without the possibility for rotating the circuits. The solution is developed in python using PuLP. The solver used is the COIN-OR Branch and Cut mixed-integer linear problem solver (CBC MIP solver).

Problem parameters As mentioned in the problem description, the problem parameters are the width of the board to place the circuits on (w), number of circuits (n), and the dimensions for each circuit where w_i is the width and h_i is the height of circuit i.

Decision variables The decision variables are the x position and y position, x and y, of the lower left corner of each circuit. For the VLSI design problem with rotation there is an additional decision variable, r, for whether or not the circuit is rotated.

Objective variable and function The objective variable (l) is the height of the fixed-width plate. The domain of l is

$$D(l) = [lower\_bound...upper\_bound],$$

where *lower_bound* is defined in function 3 and *upper_bound* is defined in function 4. The objective function is to minimize l.

Problem constraints In order to avoid overlapping circuits, for all combinations of circuits i and j where $i < j$ one of the following constraints must hold.

$$x_i + w_i \leq x_j$$
$$x_j + w_j \leq x_i$$
$$y_i + h_i \leq y_j$$

$$y_j + h_j \leq y_i$$

The first constraint prevents overlapping if circuit i is positioned left of circuit j. The second constraint prevent overlapping if circuit i is right of circuit j. If circuit i is located below circuit j the third constraint prevents overlapping. The last constraint prevents overlapping if circuit i above circuit j.

To model these constraints in MIP constants $M0, M1, M2, M3$ and binary variables $B_{i,j,0}, B_{i,j,1}, B_{i,j,2}, B_{i,j,3}$ are introduced.

$$x_i + w_i \leq x_j + M0 \times B_{i,j,0}$$

$$x_j + w_j \leq x_i + M1 \times B_{i,j,1}$$

$$y_i + h_i \leq y_j + M2 \times B_{i,j,2}$$

$$y_j + h_j \leq y_i + M3 \times B_{i,j,3}$$

If a binary variable $B_{i,j,k}$ is sat to 0 the k'th constraint must hold, and the circuits must be placed thereafter. If instead the binary variable is sat to 1 the k'th constraint does not need to hold. To ensure that all circuits are placed, at least one constraint must hold, which means that no more than three binary variables can be set to one. This is guaranteed by including the following constraint,

$$\sum_{k=0}^{3} B_{i,j,k} \leq 3.$$

In order to reduce the possibility of numerical instabilities and loss of precision the constants Mk should be as small as possible, but still large enough to allow all possible placements of circuits [1]. In y-direction the highest possible placement of a circuit is the upper bound computed in equation 4, hence $M2 = M3 = \textit{upper\_bound}$. In x-direction the circuits must be placed within the width of the board, w, therefore $M0 = M1 = w$.

To assure that no circuits exceeds the width of the plate, the following constraint is included,

$$\forall i \; x_i + w_i \leq M0,$$

where $M0 = M1 = w$.

Problem constraints rotation When a circuit is rotated the width of the circuit (w_i) is aligned in y-direction and interpreted as the height of

the circuit, and the height of the circuit (h_i) is aligned in x-direction and interpreted as the width. Hence, when to use w_i and h_i in the constraints discussed will depend on whether the circuit is rotated or not. To implement this in MIP a binary variable r is introduce for each circuit. If circuit i is rotated the variable r_i is sat to 1, otherwise $r_i = 0$. The problem constraints for rotation will be as follows,

$$
\begin{aligned}
x_i + (1 - r_i) \times w_i + r_i \times h_i &\leq x_j + M0 \times B_{i,j,0} \\
x_j + (1 - r_j) \times w_j + r_j \times h_j &\leq x_i + M1 \times B_{i,j,1} \\
y_i + (1 - r_i) \times h_i + r_i \times w_i &\leq y_j + M2 \times B_{i,j,2} \\
y_j + (1 - r_j) \times h_j + r_j \times w_j &\leq y_i + M3 \times B_{i,j,3}
\end{aligned}
$$

$$
\sum_{k=0}^{3} B_{i,j,k} \leq 3.
$$

$$
\forall i \; x_i + (1 - r_i) \times w_i + r_i \times h_i \leq M0.
$$

In order to break symmetry and reduce the search space, a constraint is added to avoid rotation of squares. A circuit i is a square if it's width is equal to it's height. If this is the case r_i is sat to 0,

$$
r_i = 0 \quad \text{if } w_i = h_i.
$$

Results

All 40 provided instances have been tested using MIP for both VLSI design problem with and without rotation. When rotation is allowed the instances have been tested with and without the breaking of symmetry. A time limit of 5 minutes have been introduced. If the solver uses more than 5 minutes to solve an instance, it will stop and mark the instance as unsolved.

Table 5 shows how many instances each implementation manage to solve without exceeding the time limit.

	Fixed orientation	Rotation	Rotation & symmetry breaking
Number of instances solved	9	5	5

Table 5: Number of instances solved out of 40 possible instances

Figure 6 displays respectively the time it took to run each instance for each of the implementations. Figure 7 displays for each implementation the number of nodes the solver visited when solving each instance. Figure 8 shows how many iterations the solver performed when solving each iteration for all implementations. The instances that timed out are not plotted in any of the the figures.

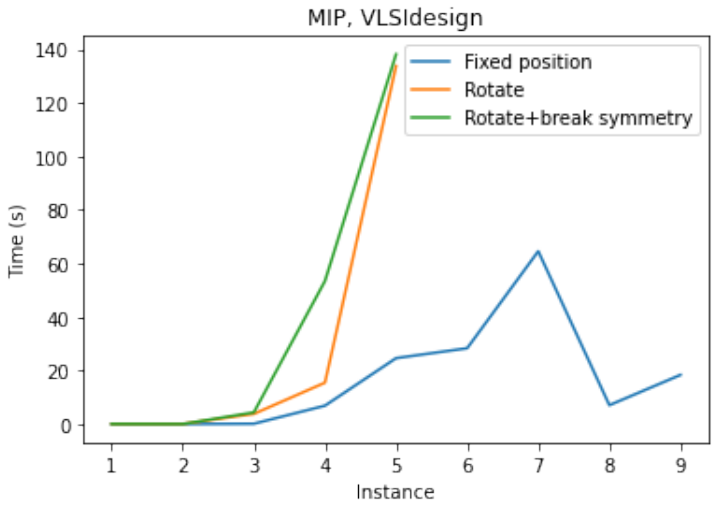

Figure 6: Solving time per instance

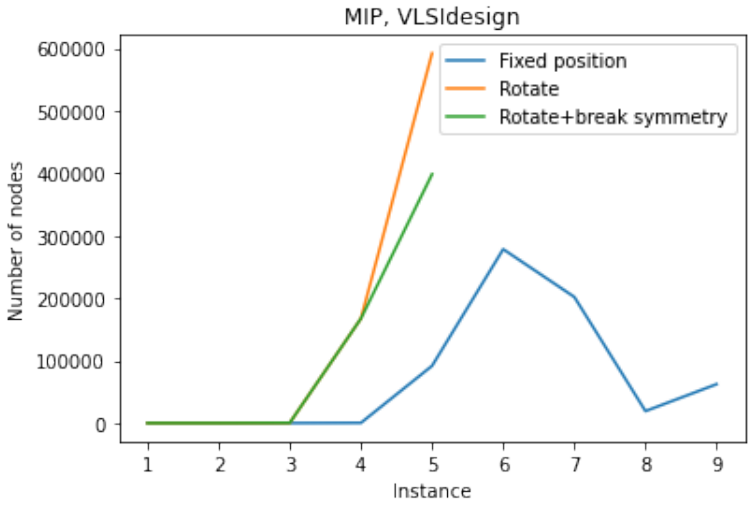

Figure 7: Number of nodes visited per instance

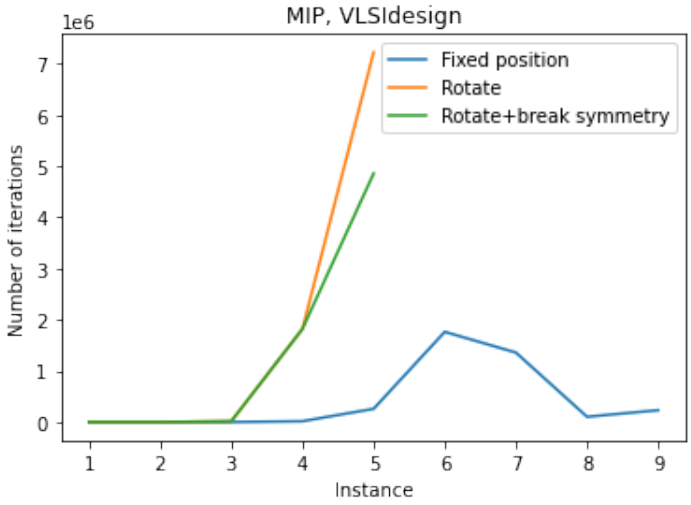

Figure 8: Number of iterations per instance

Discussion

Figure 6 demonstrates that when rotation is allowed, the solution time greatly increases, resulting in more timeouts and fewer instances solved, as seen

in table 5. Further, the number of nodes visited and number of iterations increased when rotation is allowed, as shown in figure 7 and 8. This, as well as the increase in solving time, might be due to the fact that the number of variables and possible solutions increase when the binary rotation variable is introduced.

From figure 7 and figure 8 it can be observed that breaking symmetry by fixing the orientation of squares reduces the number of nodes visited and number of iterations. However, the solving time for rotation with break of symmetry is somewhat higher than for rotation without symmetry breaking. This solving time difference could be related to the computer being utilized, where the speed of computations varies depending on other processes operating on the machine.

References

[1] M. Cococcioni and L. Fiaschi. The big-m method with the numerical infinite m. https://link.springer.com/article/10.1007/s11590-020-01644-6, 2021.

[2] Leonardo de Moura and Nikolaj Bjørner. Z3: an efficient smt solver. volume 4963, pages 337–340, 04 2008.

[3] Zeynep Kiziltan. Sat encodings. https://virtuale.unibo.it/mod/resource/view.php?id=817126, 2022.