Very Large Scale Integration
CP Model

Combinatorial Decision Making and Optimization Project Work

Francesco Ballerini
francesco.ballerini3@studio.unibo.it

AY 2020–2021

Abstract

The purpose of this work is to deploy Constraint Programming (CP) in order to solve a formulation of the Optimal Rectangle Packing problem modeling the industrial practice known as Very Large Scale Integration (VLSI): given a fixed-width plate and a list of rectangular circuits, place them on the plate so that its length is minimized. After describing our CP model, we show experimental results on sample instances highlighting performance improvements provided by implied and symmetry-breaking constraints as well as the adopted search strategy.

1 Introduction

Very Large Scale Integration (VLSI) is the process of placing circuits on a silicon plate with the goal of maximizing the resulting device portability. Such task can be modeled as follows: given a rectangular plate with fixed width and arbitrary length, rectangular circuits must be packed into the plate in such a way that the resulting plate length is the minimal one. This formalization is usually referred to in the literature as Optimal Rectangle Packing [1][2].

In the following, we model Optimal Rectangle Packing as a Constraint Optimization Problem and test a MiniZinc implementation of our formalization on a set of sample instances. Specifically, we consider two versions of the problem: one that, given a $w \times \ell$ circuit, requires to place it on the plate without changing its orientation, and the other which instead allows to position that same circuit as either a $w \times \ell$ or an $\ell \times w$ one, therefore allowing rotations.

2 Fixed-Orientation Model

In this section we will provide a detailed description of our formalization of Optimal Rectangle Packing with fixed circuit orientations.

2.1 Data

The fixed-orientation model needs to receive in input some quantities describing the specific problem instance we want to solve. Those data are:

- $plateWidth \in \mathbb{N}$: width of the plate on which circuits must be placed.
- $nCircuits \in \mathbb{N}$: number of circuits.
- $widths \in \mathbb{N}^{nCircuits}$: vector of circuit widths.
- $lengths \in \mathbb{N}^{nCircuits}$: vector of circuit lengths.

We then define some additional quantities that, although directly derivable from the instance data, will make the model constraints more clear and concise:

$$maxLength = \sum_{i=1}^{nCircuits} lengths[i]$$
$$minLength = \left\lceil \frac{\sum_{i=1}^{nCircuits} widths[i] \cdot lengths[i]}{plateWidth} \right\rceil$$

$maxLengtht$ is the plate length resulting from circuits being all stacked on top of each other; we will treat it as an upper bound for the plate length. $minLength$ is instead the plate length obtained if circuits span over the entire plate width with no empty space being left between them—computed by redistributing the total area of the circuits "homogeneously" throughout the plate; we will use it when we need to identify a lower region of the plate.

2.2 Variables

The variables of fixed-orientation model encode the circuit placements on the plate:

- $xs \in \mathbb{N}^{nCircuits}$: vector of horizontal coordinates of the bottom-left corners of the circuits.
- $ys \in \mathbb{N}^{nCircuits}$: vector of vertical coordinates of the bottom-left corners of the circuits.

The origin of the coordinate system is the bottom-left corner of the plate.

2.3 Main Constraints

First, we impose the global constraint

$$\textsc{Diffn}(xs, ys, widths, lengths)$$

which constrains rectangles i with origins $(xs[i], ys[i])$ and sizes $(widths[i], lengths[i])$ to be non-overlapping. Then, we still need to prevent circuits from exceeding the plate width, hence the constraint

$$\max\{xs[i] + widths[i] \mid i = 1, \ldots, nCircuits\} \leq plateWidth$$

2.4 Implied Constraints

In our terminology, we call *implied* those constraints that are semantically redundant but computationally significant, as they reduce the search space while keeping the set of solutions unchanged.

An alternative interpretation, which we will adopt to introduce additional constraints, is to see Rectangle Packing as two scheduling problems—with time flowing in the x and y-axis direction, respectively—each sharing a resource among its tasks [3]. Let P_x and P_y denote those two scheduling problems. P_x requires to schedule $nCircuit$ tasks with start times equal to circuit x-coordinates, durations equal to circuit widths, resource requirements equal to circuit lengths, and resource capacity equal to the plate length. Analogously, P_y requires to schedule $nCircuit$ tasks with start times equal to circuit y-coordinates, durations equal to circuit lengths, resource requirements equal to circuit widths, and resource capacity equal to the plate width. In other words, P_x has time dimension along the x-axis and resource dimension along the y-axis, while the opposite holds for P_y. With this view in mind, we can impose two Cumulative global constraints encoding P_x and P_y, respectively:

$$\textsc{Cumulative}(xs, widths, lenghts, maxLength)$$
$$\textsc{Cumulative}(ys, lengths, widths, plateWidth)$$

Notice that, since the plate length is an unknown quantity of our problem—it is what we want to minimize—we are forced to weaken the P_x requirements by setting the resource capacity to an upper bound of the plate length, i.e. $maxLength$.

2.5 Symmetry-Breaking Constraints

Given a circuit configuration, we can flip the plate over the x or y axis and obtain an equivalent solution. We thus leverage this observation to reduce the search space by forcing the circuit with the biggest area to be placed in the bottom-left region of the plate [3]:

$$i^* = \operatorname{argmax}\{widths[i] \cdot lengths[i] \mid i = 1, \ldots, nCircuits\}$$

$$xs[i^*] \leq \left\lfloor \frac{plateWidth - widths[i^*]}{2} \right\rfloor$$
$$ys[i^*] \leq \left\lfloor \frac{minLength - lengths[i^*]}{2} \right\rfloor$$

The rationale behind the choice of fixing the position of the biggest-area circuit is that, by taking up the most space on the plate, it is the most likely to have an impact on the positioning of subsequent circuits [2].

2.6 Objective

As we already anticipated, the goal of our model is to minimize the plate length reached by the chosen circuit placements:

$$\min \max \{ys[i] + lengths[i] \mid i = 1, \ldots, nCircuits\}$$

3 Rotation Model

A possible variation on the fixed-orientation model of Section 2 works as follows: if the input instance defines a circuit as a $w \times \ell$ rectangle, then it can be placed on the plate as either a $w \times \ell$ or an $\ell \times w$ circuit, i.e. either in its original or rotated version.

3.1 Data

In order to deal with rotations, we must augment the data representation introduced in Section 2.1. Specifically, our data definition will be heavily affected by the signature requirements of the Geost global constraint—see Section 3.3—which is meant to handle arbitrary shapes given by the composition of a set of rectangles [4]. In our case, a shape will trivially consist of a single rectangle, either in its original or rotated form; in other words, we are only interested in a specific subset of the kind of problems Geost can be applied to. This will result is a somewhat cumbersome and overcomplicated data specification, which is however the price to pay in order to enjoy the benefits of global constraints when generalizing Rectangle Packing by allowing rotations:

- $plateWidth \in \mathbb{N}$: width of the plate on which circuits must be placed.
- $nCircuits \in \mathbb{N}$: number of circuits.
- $nShapes \in \mathbb{N}$: each $w \times \ell$ circuit can be placed on the plate as two *shapes*—either a $w \times \ell$ rectangle or an $\ell \times w$ one; $nShapes$ is the number of such shapes. To be precise, while a rectangular circuit is mapped to two shapes, a square one does not have a meaningful rotated counterpart, hence $nCircuits \leq nShapes \leq 2 \cdot nCircuits$.

- $widths \in \mathbb{N}^{nCircuits}$: vector of widths of the (unrotated) circuits.
- $lengths \in \mathbb{N}^{nCircuits}$: vector of lengths of the (unrotated) circuits.
- $sizes \in \mathbb{N}^{nShapes \times 2}$: matrix containing, for each $w \times \ell$ circuit, both (w, ℓ) and (ℓ, w) as rows.
- $offsets \in \mathbb{N}^{nShapes \times 2}$: matrix needed by the Geost global constraint [4], which, however, is meaningless in our problem formulation; all its entries are therefore set to zero.
- $shapes \in \mathcal{P}(\mathbb{N})^{nShapes}$: vector of sets[1], one for each shape, where each set contains an unique integer identifier of the corresponding shape.
- $validShapes \in \mathcal{P}(\mathbb{N})^{nCircuits}$: vector of sets, one for each circuit, where each set contains the identifiers of shapes corresponding to the same circuit, i.e. one identifier for squares and two for rectangles.

An example of model data for a given instance is shown in Figure 1.

Furthermore, similarly to what we did for the fixed-orientation model, we define the auxiliary quantities

$$maxLength = \sum_{i=1}^{nCircuits} \max\{widths[i], lengths[i]\}$$
$$minLength = \left\lceil \frac{\sum_{i=1}^{nCircuits} widths[i] \cdot lengths[i]}{plateWidth} \right\rceil$$

where $maxLength$ is the plate length resulting from circuits being all stacked on top of each other and oriented so that their longer edge is the vertical one, while $minLength$ has the same meaning as explained in Section 2.1.

3.2 Variables

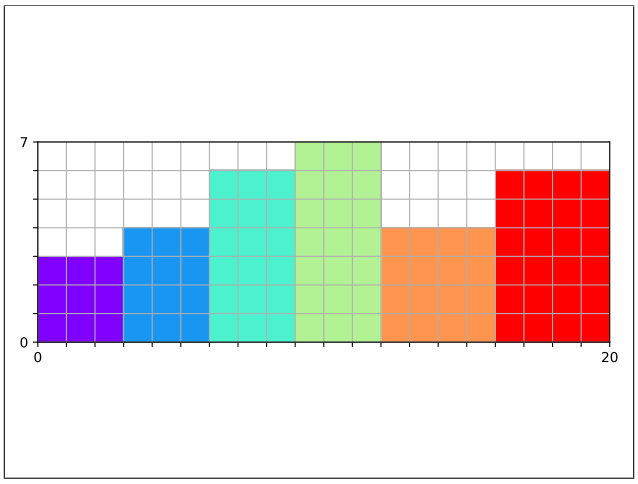
Geost's signature will also require the following variables:

- $positions \in \mathbb{N}^{nCircuits \times 2}$: for each circuit i, $positions[i] = (x_i, y_i)$, where x_i and y_i are the horizontal and vertical coordinates of the bottom-left corner of i, respectively.
- $actualShapes \in \mathbb{N}^{nCircuits}$: for each circuit i, $actualShapes[i]$ is the identifier of the shape associated to i, i.e. either its original or rotated version.

However, in order to simplify the formulation of the other constraints, we also define the auxiliary variables

$$xs \in \mathbb{N}^{nCircuits}, \text{ where } xs[i] = positions[i, 1]$$

[1] $\mathcal{P}(\mathbb{N})$ is the powerset of $\mathbb{N}$.



(a) Instance 3

```
plate_width = 10;
n_circuits = 6;
n_shapes = 10;
widths = [3, 3, 3, 3, 4, 4];
lengths = [3, 4, 6, 7, 4, 6];
sizes = [| 3, 3 | 3, 4 | 4, 3 | 3, 6 | 6, 3 | 3, 7 | 7, 3 | 4, 4 | 4, 6 | 6, 4 |];
offsets = [| 0, 0 | 0, 0 | 0, 0 | 0, 0 | 0, 0 | 0, 0 | 0, 0 | 0, 0 | 0, 0 | 0, 0 |];
shapes = [{1}, {2}, {3}, {4}, {5}, {6}, {7}, {8}, {9}, {10}];
valid_shapes = [{1}, {2, 3}, {4, 5}, {6, 7}, {8}, {9, 10}];
```

(b) `.dzn` file

Figure 1: A problem instance and its corresponding MiniZinc data file for the rotation model.

$$
\begin{aligned}
ys &\in \mathbb{N}^{nCircuits}, \text{ where } ys[i] = positions[i, 2] \\
ws &\in \mathbb{N}^{nCircuits}, \text{ where } ws[i] = sizes[actualShapes[i], 1] \\
\ell s &\in \mathbb{N}^{nCircuits}, \text{ where } \ell s[i] = sizes[actualShapes[i], 2]
\end{aligned}
$$

ws and ℓs are therefore, respectively, the widths and lengths of the circuits in their chosen orientation, i.e.

$$
(ws[i], \ell s[i]) = \begin{cases} (lengths[i], widths[i]) & \text{if circuit } i \text{ is rotated} \\ (widths[i], lengths[i]) & \text{otherwise} \end{cases}
$$

whereas xs and ys have the same meaning as described in Section 2.2.

3.3 Main Constraints

A more general version of the DIFFN global constraint we saw in section 2.3 is the GEOST global constraint, which enforces k-dimensional objects to be non-overlapping. In our case, $k = 2$ and

$$\text{GEOST}(2, sizes, offsets, shapes, positions, actualShapes)$$

selects circuit positions and orientations so that they do not overlap. Furthermore, as we did for our fixed-orientation model, we require circuits to not exceed the plate width:

$$\max \{xs[i] + ws[i] \mid i = 1, \ldots, nCircuits\} \leq plateWidth$$

Finally, for each circuit, we must enforce its orientation to be one of two possible choices:

$$\forall i = 1, \ldots, nCircuits \quad actualShapes[i] \in validShapes[i]$$

3.4 Implied Constraints

As already discussed in Section 2.4, Rectangle Packing can be seen as two scheduling problems along the two axes. By leveraging this observation, the same reasoning we applied to the fixed-orientation model leads us to the following implied constraints:

$$\text{CUMULATIVE}(xs, ws, \ell s, maxLength)$$
$$\text{CUMULATIVE}(ys, \ell s, ws, plateWidth)$$

3.5 Symmetry-Breaking Constraints

In order to achieve a partial pruning of symmetries, we adopt the same strategy as described in Section 2.5, hence the following constraints:

$$i^* = \text{argmax} \{widths[i] \cdot lengths[i] \mid i = 1, \ldots, nCircuits\}$$
$$xs[i^*] \leq \left\lfloor \frac{plateWidth - ws[i^*]}{2} \right\rfloor$$
$$ys[i^*] \leq \left\lfloor \frac{minLength - \ell s[i^*]}{2} \right\rfloor$$

3.6 Objective

The introduction of rotations in our problem formulation does not change the semantics of its objective function; the objective of the rotation model is therefore

$$\min \max \{ys[i] + \ell s[i] \mid i = 1, \ldots, nCircuits\}$$

Model	# Solved (< 5 min)
Base	5
Implied	15
Symmetry	16

Table 1: Effect of implied and symmetry-breaking constraints on the number of instances solved under 5 minutes by Gecode when run on the fixed-orientation model with Input-Order variable ordering and Indomain-Min value ordering.

4 Results

In this section we will show experimental results obtained by running a MiniZinc implementation of the models described in Sections 2 and 3 on a MacBook Pro with CPU Intel Core i5 quad-core at 1.4 GHz and 16 GB of RAM. The models are tested on a set of 40 sample instances and the solving process is aborted after 5 minutes: if optimality has not yet been proven by then, the instance is considered unsolved.

4.1 Implied and Symmetry-Breaking Constraints

First, in order to test the effect of implied and symmetry-breaking constraints on the fixed-orientation model, we compare the performance of the following increasingly constrained formalizations:

- Base: fixed-orientation model including only the main constraints of Section 2.3.
- Implied: fixed-orientation model containing both the main constraints of Section 2.3 and the implied constraints of Section 2.4.
- Symmetry: fixed-orientation model including the main constraints of Section 2.3, the implied constraints of Section 2.4, and the symmetry-breaking constraints of Section 2.5.

Furthermore, to minimize the impact of the search strategy on the comparison, we run the three models with the Input-Order variable-ordering and Indomain-Min value-ordering heuristics of the Gecode solver, i.e. variables in xs and ys are selected according to the order they are defined in, whereas their values are picked in increasing order. Results are shown in Tables 1 and 3: while the introduction of implied constraints has a very noticeable effect, which becomes apparent even just by looking at the number of instances solved within the time limit, the further addition of symmetry-breaking constraints provides more nuanced but consistent improvements, which can be observed in the outcome of the search-space exploration revealed by the solving statistics.

Similarly, we compare the results given by the following increasingly constrained variations of the rotation model:

Model	# Solved (< 5 min)
Base-Rot	3
Implied-Rot	6
Symmetry-Rot	6

Table 2: Effect of implied and symmetry-breaking constraints on the number of instances solved under 5 minutes by Gecode when run on the rotation model with Input-Order variable ordering and Indomain-Min value ordering.

- Base-Rot: rotation model including only the main constraints of Section 3.3.
- Implied-Rot: rotation model containing both the main constraints of Section 3.3 and the implied constraints of Section 3.4.
- Symmetry-Rot: rotation model including the main constraints of Section 3.3, the implied constraints of Section 3.4, and the symmetry-breaking constraints of Section 3.5.

Again, the comparison is made by running Gecode with Input-Order variable ordering and Indomain-Min value ordering. The results in Tables 2 and 4 show a behavior that is comparable to that of the fixed-rotation model: the introduction of implied constraints increases the number of instances solved before the timeout, whereas the further addition of symmetry-breaking constraints leads to more subtle improvements which can be noticed when looking at the solving statistics.

However, a striking difference between fixed-orientation and rotation model—revealed already by these simple tests—is how much more computational effort the latter needs to solve a given instance. An intuitive explanation of this experimental evidence is that, by allowing circuits to be arranged on the plate with two alternative orientations, we are also growing the set of positions they can occupy without violating the constraints—i.e. the legal values of our model variables—therefore extending the search-space region that needs to be explored in order to find the optimal solution.

4.2 Solver and Search Strategy

Once shown that implied and symmetry-breaking constraints enable a more efficient search-space exploration—see Section 4.1—we look for the solver–search combination giving the best results when applied to the Symmetry model. Results are shown in Table 5: the highest number of solved instances is obtained with Gecode when variables in xs and ys are selected in decreasing order of circuits areas, i.e. $xs[i], ys[i]$ are picked before $xs[j], ys[j]$ if and only if

$$widths[i] \cdot lengths[i] > widths[j] \cdot lengths[j]$$

and their selected value is always the smallest available one—that is, Ordered-By-Area+ Indomain-Min. The rationale behind an ordering by areas, as we already pointed out in Section 2.5, is that bigger circuits are likely to have a stronger impact on subsequent placements, hence our Ordered-By-Area heuristic is essentially a fail-first principle applied to Rectangle Packing [2].

Random-value selection, on the other hand, leads to poor experimental results when performed by Gecode on the model: outputs as those shown in Figure 2 suggest that, being *maxLength* quite a generous upper bound for the optimal plate length, randomly selecting values for variables in ys leads to relatively high y-coordinates, which in turn are likely to result in the exploration of suboptimal circuit configurations. For the same reason, Indomain-Min turns out to be a good value-ordering heuristic for our problem formulation, instead of merely acting as a baseline. That is why, whenever we want to introduce a form of randomness in subsequent experiments, we restrict the random-value selection to xs alone—see Table 5.

The solver–search combination giving the best results on Symmetry-Rot is instead Chuffed with its default search, as shown in Table 6. However, the best search strategy on Symmetry-Rot only solves half the amount of instances (13) solved by the best search strategy on Symmetry (26) before the 5-minute timeout; more generally, as one can see by comparing Tables 5 and 6, given any solver–search combination among the ones we tested, Symmetry solves more instances than Symmetry-Rot with that same combination. This confirms our initial intuition, discussed in Section 4.1, suggesting that Optimal Rectangle Packing with rotations is intrinsically a harder problem than its fixed-orientation counterpart.

As a final note, by visually inspecting the solutions produced in output when the solving process is aborted, we noticed that sometimes the rotation model exceeds the timeout even though an optimal solution has already been found, suggesting that the model got stuck in proving its optimality—see Figure 3. Again, this behavior agrees with the assumption that finding an optimal solution when rotations are allowed requires the inspection of a larger portion of the search space than the one that needs to be explored in the fixed-orientation case.

5 Conclusions

We modeled Optimal Rectangle Packing—both with and without rotations—as a Constraint Optimization Problem and tested a MiniZinc implementation of our formalization on a set of 40 sample instances. Specifically, we performed experiments justifying the addition of implied and symmetry-breaking constraints and then fine-tuned the search strategy in order to achieve better results. When setting the maximum execution time to 5 minutes, our best fixed-orientation and rotation models were able to prove optimality for 26 and 13 instances, respectively.

Instance	Model	Time	Nodes	Failures
1	Base	121ms	25	9
	Implied	102ms	25	9
	Symmetry	172ms	4	1
2	Base	114ms	36	13
	Implied	114ms	36	13
	Symmetry	126ms	11	4
3	Base	121ms	3976	1981
	Implied	121ms	163	74
	Symmetry	112ms	28	12
4	Base	1s 943ms	1208065	604024
	Implied	120ms	360	171
	Symmetry	108ms	123	57
5	Base	42s 419ms	23131481	11565738
	Implied	130ms	112	53
	Symmetry	119ms	24	9
6	Implied	142ms	3588	1785
	Symmetry	144ms	2010	992
7	Implied	132ms	1989	987
	Symmetry	117ms	1726	852
8	Implied	116ms	335	156
	Symmetry	123ms	96	41
9	Implied	130ms	2977	1473
	Symmetry	117ms	3070	1522
10	Implied	298ms	30893	15437
	Symmetry	291ms	29764	14871
12	Implied	25s 262ms	5961600	2980773
	Symmetry	24s 719ms	5945676	2972823
13	Implied	11s 761ms	2944944	1472450
	Symmetry	4s 768ms	1179632	589798
14	Implied	1m 48s	23946155	11973064
	Symmetry	8s 706ms	1878311	939140
15	Implied	1s 205ms	193381	96657
	Symmetry	283ms	28902	14416
17	Symmetry	3m 54s	43800131	21900039
18	Implied	16s 296ms	2526342	1263133
	Symmetry	15s 468ms	2408712	1204342

Table 3: Solving statistics of the three increasingly constrained fixed-orientation models when solved under 5 minutes by Gecode with Input-Order variable ordering and Indomain-Min value ordering. Models are shown below the second column only when able to solve the instance within the time limit. Notice that instance 9 is the only one for which Symmetry does not outperform Implied in terms of nodes and failures.

Instance	Model	Time	Nodes	Failures
1	Base-Rot	243ms	4264	2131
	Implied-Rot	155ms	14	14
	Symmetry-Rot	128ms	7	1
2	Base-Rot	1s 138ms	50803	25398
	Implied	278ms	57	23
	Symmetry	282ms	41	18
3	Base-Rot	4m 16s	9866051	4933023
	Implied-Rot	494ms	1906	950
	Symmetry-Rot	445ms	752	372
4	Implied-Rot	12s 948ms	221507	110750
	Symmetry-Rot	6s 684ms	96452	48224
5	Implied-Rot	1m 10s	374682	187337
	Symmetry-Rot	56s 144ms	263779	131886
6	Implied-Rot	2m 32s	577046	288519
	Symmetry-Rot	1m 27s	306909	153453

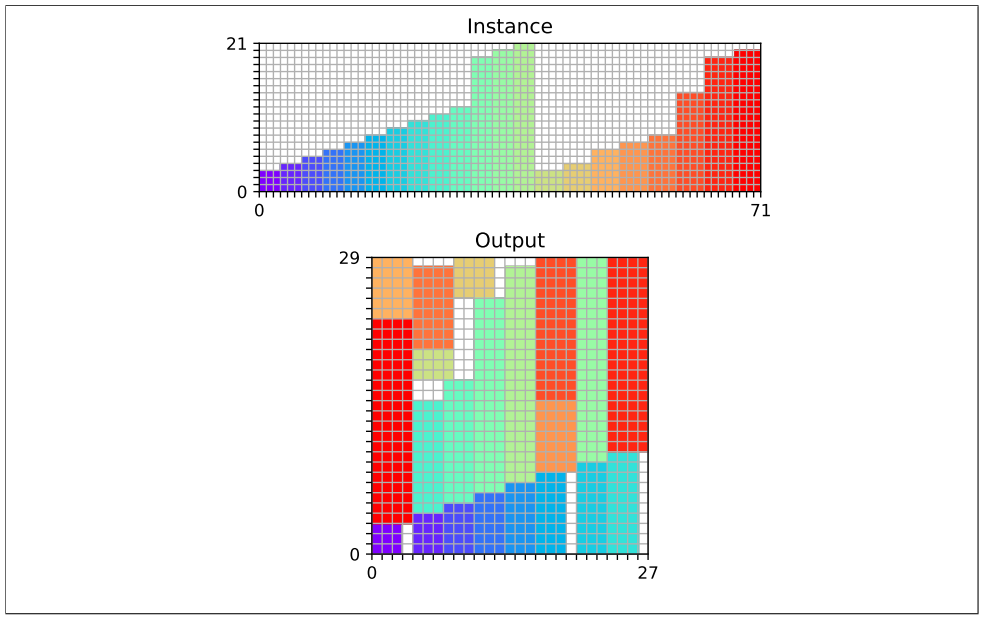
Table 4: Solving statistics of the three increasingly constrained rotation models when solved under 5 minutes by Gecode with Input-Order variable ordering and Indomain-Min value ordering. Models are shown below the second column only when able to solve the instance within the time limit. Notice that, although Implied-Rot and Symmetry-Rot solve the same number of instances before the timeout, Symmetry-Rot always outperforms Implied-Rot in terms of nodes and failures.

Solver	Variable Ordering	Value Ordering	Restart	# Solved (< 5 min)
Gecode	Input-Order	Indomain-Min(xs, ys)	None	16
	Input-Order	Indomain-Random(xs, ys)	None	11
	Dom-W-Deg	Indomain-Min(xs, ys)	None	16
	Dom-W-Deg	Indomain-Random(xs, ys)	None	14
	Dom-W-Deg	Indomain-Random(xs) + Indomain-Min(ys)	Luby(1000)	18
	Dom-W-Deg	Indomain-Random(xs) + Indomain-Min(ys)	Luby(1000) + Lns(80)	13
	Ordered-By-Area	Indomain-Min(xs, ys)	None	**26**
	Ordered-By-Area	Indomain-Random(xs) + Indomain-Min(ys)	Luby(1000)	9
	Ordered-By-Area	Indomain-Random(xs) + Indomain-Min(ys)	Luby(1000) + Lns(80)	14
Chuffed	Default	Default	Default	24
	Ordered-By-Area	Indomain-Min(xs, ys)	Default	23

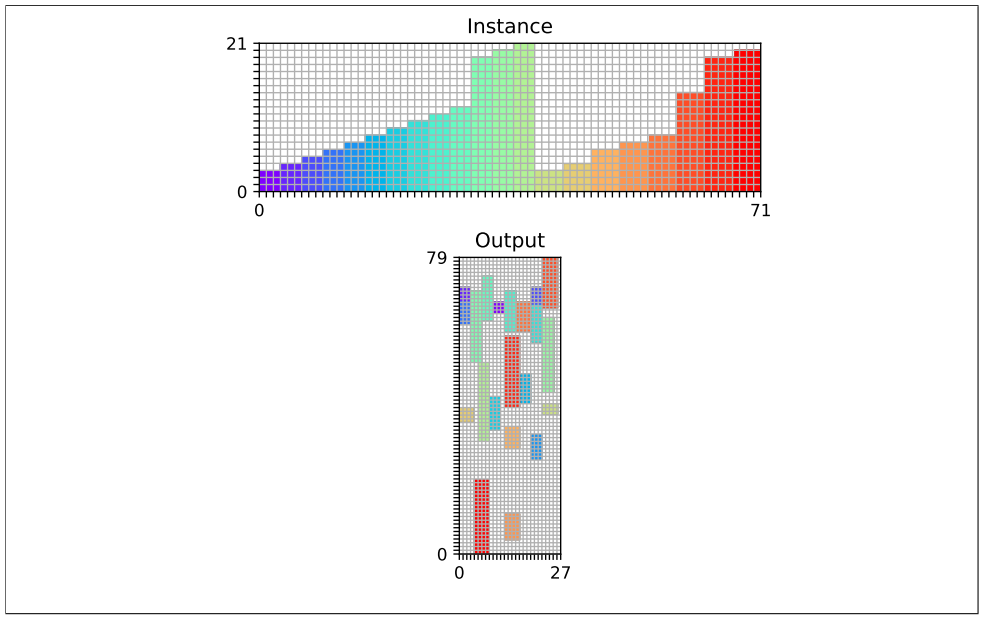
Table 5: Number of instances solved within the time limit of 5 minutes by Symmetry with different solver–search combinations. Input-Order, Dom-W-Deg, Indomain-Min, Indomain-Random, and Luby refer to predefined MiniZinc search annotations. Ordered-By-Area denotes variable selection in decreasing order of circuit areas, whereas Lns—which stands for Large Neighborhood Search—corresponds to Gecode's Relax-And-Reconstruct. When, for example, Indomain-Min(xs, ys) is reported in the table, it means applying the Indomain-Min value-ordering heuristic to both variables xs and ys.

Solver	Variable Ordering	Value Ordering	Restart	# Solved (< 5 min)
Gecode	Input-Order	Indomain-Min(xs, ys)	None	6
	Input-Order	Indomain-Random(xs, ys)	None	5
	Dom-W-Deg	Indomain-Min(xs, ys)	None	6
	Dom-W-Deg	Indomain-Random(xs, ys)	None	4
	Dom-W-Deg	Indomain-Random(xs) + Indomain-Min(ys)	Luby(1000)	4
	Dom-W-Deg	Indomain-Random(xs) + Indomain-Min(ys)	Luby(1000) + Lns(80)	1
	Ordered-By-Area	Indomain-Min(xs, ys)	None	7
	Ordered-By-Area	Indomain-Random(xs) + Indomain-Min(ys)	Luby(1000)	6
	Ordered-By-Area	Indomain-Random(xs) + Indomain-Min(ys)	Luby(1000) + Lns(80)	1
Chuffed	Default	Default	Default	**13**
	Ordered-By-Area	Indomain-Min(xs, ys)	Default	10

Table 6: Number of instances solved within the time limit of 5 minutes by Symmetry-Rot with different solver–search combinations. Input-Order, Dom-W-Deg, Indomain-Min, Indomain-Random, and Luby refer to predefined MiniZinc search annotations. Ordered-By-Area denotes variable selection in decreasing order of circuit areas, whereas Lns—which stands for Large Neighborhood Search—corresponds to Gecode's Relax-And-Reconstruct. When, for example, Indomain-Min(xs, ys) is reported in the table, it means applying the Indomain-Min value-ordering heuristic to both variables xs and ys.

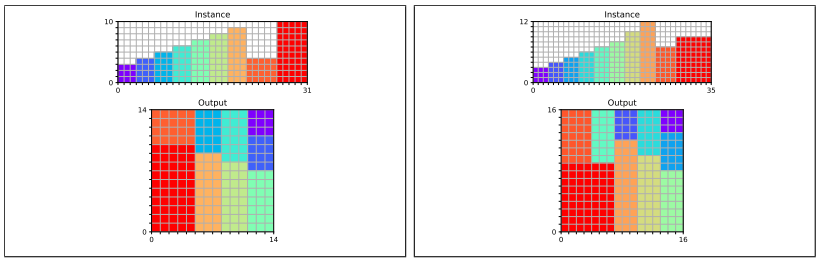


(a) Indomain-Min



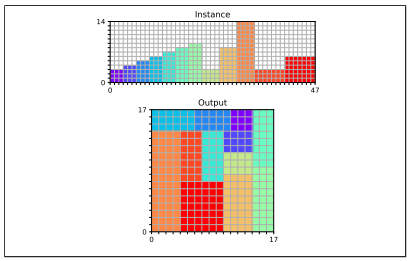
(b) Indomain-Random

Figure 2: Output of instance 20 when Gecode is run on Symmetry with Input-Order variable ordering and either Indomain-Min or Indomain-Random value ordering. Those shown above are the suboptimal solutions produced by the solver when stopped at the 5-minute mark—the optimal solution has plate length = 27. Notice how Indomain-Random leads to the exploration of circuit configurations that are very far from the optimal one.

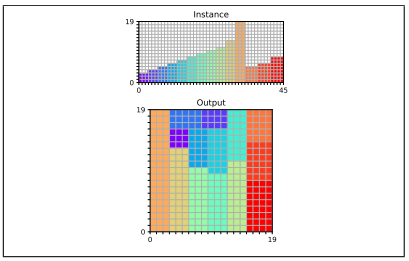


(a) Instance 7

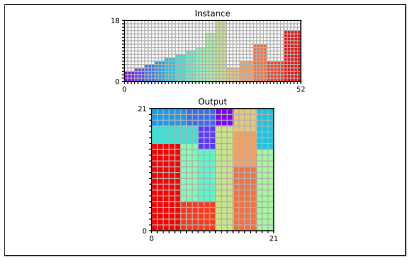
(b) Instance 9



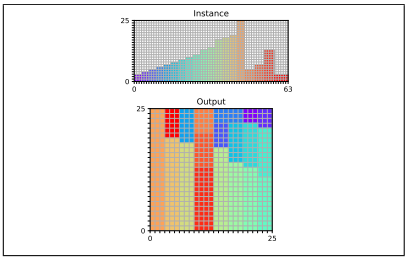
(c) Instance 10



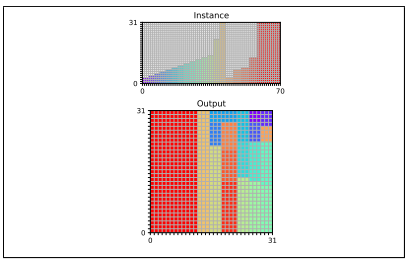
(d) Instance 12



(e) Instance 14



(f) Instance 18



(g) Instance 24

Figure 3: Output of Symmetry-Rot with Ordered-By-Area + Indomain-Min search strategy on instances where the timeout is exceeded but, by visual inspection, we can see that the solver (Gecode) has already found an optimal solution—without, however, being able to prove it before the 5-minute mark. For this reason, we did not include the above instances in the count of the corresponding model in Table 6; nonetheless, they can still provide an insight on the effectiveness of the Ordered-By-Area heuristic when applied to the rotation model.

References

[1] Richard E. Korf. Optimal Rectangle Packing: Initial Results. ICAPS 2003.

[2] Michael D. Moffitt and Martha E. Pollack. Optimal Rectangle Packing: A Meta-CSP Approach. ICAPS 2006.

[3] Helmut Simonis and Barry O'Sullivan. Using Global Constraints for Rectangle Packing. BPPC 2008.

[4] Understanding the input format of Minizinc's geost constraint (Stack Overflow thread). `https://stackoverflow.com/a/60654136`