Very Large Scale Integration
SMT 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 Satisfiability Modulo Theories (SMT) 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 SMT model, we show experimental results on sample instances highlighting performance improvements provided by implied and symmetry-breaking constraints.

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 Satisfiability Modulo Theories problem and test a Z3Py 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:

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

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 the 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 enforce circuits to be non-overlapping:

$$\begin{aligned}
&(xs[i] + widths[i] \leq xs[j]) \vee \\
&(xs[j] + widths[j] \leq xs[i]) \vee \\
&(ys[i] + lengths[i] \leq ys[j]) \vee \\
&(ys[j] + lengths[j] \leq ys[i]) \quad \forall\, 1 \leq i < j \leq nCircuits
\end{aligned}$$

That is, given any two circuits i and j with $i \neq j$, one out of four options holds: i is on the left of j, i is on the right of j, i is below j, or i is above j [2]. Furthermore, we must prevent circuits from exceeding the plate width, hence the following constraint:

$$xs[i] + widths[i] \leq plateWidth \quad \forall i = 1, \ldots, nCircuits$$

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 that in mind, let us define the constraint we need to model the above scenario: given n tasks with start times $s \in \mathbb{N}^n$, durations $d \in \mathbb{N}^n$, resource requirements $r \in \mathbb{N}^n$, and resource capacity $c \in \mathbb{N}$, let *cumulative* be defined as

$$cumulative(s, d, r, c) \iff \sum_{i:\ s[i] \leq t \leq s[i]+d[i]} r[i] \leq c \quad \forall t = 0, \ldots, \textstyle\sum_{j=1}^{n} d[j] \tag{1}$$

i.e. at any time unit t, the resource capacity is not exceeded [4]. The two constraints encoding P_x and P_y, respectively, can now be expressed as

$$\begin{gathered}
cumulative(xs, widths, lenghts, maxLength) \\
cumulative(ys, lengths, widths, plateWidth)
\end{gathered}$$

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]:

$$\begin{aligned} i^* &= \operatorname{argmax}\{\mathit{widths}[i] \cdot \mathit{lengths}[i] \mid i = 1, \dots, \mathit{nCircuits}\} \\ \mathit{xs}[i^*] &\leq \left\lfloor \frac{\mathit{plateWidth} - \mathit{widths}[i^*]}{2} \right\rfloor \\ \mathit{ys}[i^*] &\leq \left\lfloor \frac{\mathit{minLength} - \mathit{lengths}[i^*]}{2} \right\rfloor \end{aligned}$$

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:

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

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

The data of the rotation model are the same as those introduced in Section 2.1:

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

- $\mathit{lengths} \in \mathbb{N}^{nCircuits}$: vector of lengths of the (unrotated) circuits.

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

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

where $\mathit{maxLength}$ 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 $\mathit{minLength}$ has the same meaning as explained in Section 2.1.

3.2 Variables

In addition to the variables of the fixed-orientation model

$$\begin{aligned} xs &\in \mathbb{N}^{nCircuits} \\ ys &\in \mathbb{N}^{nCircuits} \end{aligned}$$

encoding, respectively, the horizontal and vertical coordinates of the bottom-left circuit corners, we also need some variables storing information about circuit rotations:

$$\begin{aligned} \mathit{rot} &\in \{\textsc{True}, \textsc{False}\}^{nCircuits} \\ \mathit{widthsRot} &\in \mathbb{N}^{nCircuits} \\ \mathit{lengthsRot} &\in \mathbb{N}^{nCircuits} \end{aligned}$$

whose intended meaning is that $\forall i = 1, \ldots, nCircuits$

$$(\mathit{rot}[i], \mathit{widthsRot}[i], \mathit{lengthsRot}[i]) = \begin{cases} (\textsc{True}, \mathit{lengths}[i], \mathit{widths}[i]) & \text{if circuit } i \text{ is rotated} \\ (\textsc{False}, \mathit{widths}[i], \mathit{lengths}[i]) & \text{otherwise} \end{cases}$$

i.e. $\mathit{rot}[i]$ is a Boolean variable encoding whether circuit i is rotated or not, whereas $\mathit{widthsRot}[i]$ and $\mathit{lengthsRot}[i]$ store the dimensions of i according to its orientation.

3.3 Main Constraints

First, we encode the notion that, if a circuit is a square, then it has no rotated counterpart, as its rotation is undistinguishable from its original orientation:

$$(\mathit{widths}[i] = \mathit{lengths}[i]) \implies \neg\, \mathit{rot}[i] \quad \forall i = 1, \ldots, nCircuits$$

Then, we define the meaning of variables *widthsRot* and *lengthsRot*, as discussed in Section 3.2:

$$\begin{aligned}\big(rot[i] &\implies (widthsRot[i] = lengths[i])\big) \land \\ \big(rot[i] &\implies (lengthsRot[i] = widths[i])\big) \land \\ \big(\neg\, rot[i] &\implies (widthsRot[i] = widths[i])\big) \land \\ \big(\neg\, rot[i] &\implies (lengthsRot[i] = lengths[i])\big) \quad \forall i = 1, \dots, nCircuits\end{aligned}$$

Finally, as we did for our fixed-orientation model in Section 2.3, we enforce circuits to be non-overlapping:

$$\begin{aligned}&(xs[i] + widthsRot[i] \leq xs[j]) \lor \\ &(xs[j] + widthsRot[j] \leq xs[i]) \lor \\ &(ys[i] + lengthsRot[i] \leq ys[j]) \lor \\ &(ys[j] + lengthsRot[j] \leq ys[i]) \quad \forall\, 1 \leq i < j \leq nCircuits\end{aligned}$$

and require them to not exceed the plate width:

$$xs[i] + widthsRot[i] \leq plateWidth \quad \forall i = 1, \dots, nCircuits$$

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:

$$\begin{aligned}&cumulative(xs, ws, \ell s, maxLength) \\ &cumulative(ys, \ell s, ws, plateWidth)\end{aligned}$$

where *cumulative* is defined in Equation 1.

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:

$$\begin{aligned}i^* &= \text{argmax}\,\{widths[i] \cdot lengths[i] \mid i = 1, \dots, nCircuits\} \\ xs[i^*] &\leq \left\lfloor \frac{plateWidth - widthsRot[i^*]}{2} \right\rfloor \\ ys[i^*] &\leq \left\lfloor \frac{minLength - lengthsRot[i^*]}{2} \right\rfloor\end{aligned}$$

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

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

4 Results

In this section we will show experimental results obtained by running a Z3Py 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.

Results are shown in Tables 1, 3, 4 and in Figure 1: within the time-limit restriction, the introduction of implied constraints allows to solve 4 instances more than the 20 solved by the Base model, whereas symmetry-breaking constraints increase the number of solved instances only by 1, reaching 25 instances total—see Table 1. When comparing solving time, Figure 1a shows that, on average, the introduction of implied constraints slows down the execution, which then becomes slightly faster when adding symmetry-breaking constraints, still without reaching back the original lower solving time of the Base model. Finally, Figure 1b shows that the clear drop in conflicts obtained by going from Base to Implied is not matched when comparing Implied to Symmetry, although a slight improvement can still be noticed. However, a more systematic inspection of Tables 3 and 4 reveals that Symmetry produces less conflicts than Implied in 16 out of the 24 instances solved under 5 minutes by both models, therefore suggesting that symmetry-breaking constraints indeed lead to a more efficient search-space exploration.

Model	# Solved ($< 5\,\text{min}$)
Base	20
Implied	24
Symmetry	**25**

Table 1: Effect of implied and symmetry-breaking constraints on the number of instances solved under 5 minutes by the Z3 solver when run on the fixed-orientation model.

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

- 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.

Results are shown in Tables 2, 5 and in Figure 2: in this case, implied and symmetry-breaking constraints do not affect significantly the number of instances solved within the time limit; however, as shown in Figure 1b, the number of conflicts tends to drop as constraints are being added, whereas solving time, as already observed for the fixed-orientation model, is on average the lowest in Base-Rot and the highest in Implied-Rot—see Figure 1a.

As a final consideration, a striking difference between fixed-orientation and rotation model—independent of implied and symmetry-breaking constraints—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.

Figure 3 shows a comparison between solutions of Symmetry and Symmetry-Rot on the same instances.

5 Conclusions

We modeled Optimal Rectangle Packing—both with and without rotations—as a Satisfiability Modulo Theories problem and tested a Z3Py implementation of our formalization on a set of 40 sample instances. Specifically, we performed experiments justifying the addition of implied and symmetry-breaking constraints. When setting the maximum execution time

Model	# Solved (< 5 min)
Base-Rot	9
Implied-Rot	9
Symmetry-Rot	**10**

Table 2: Effect of implied and symmetry-breaking constraints on the number of instances solved under 5 minutes by the Z3 solver when run on the rotation model.

to 5 minutes, our best fixed-orientation and rotation models were able to prove optimality for 25 and 10 instances, respectively.

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] Cumulative constraint. Mathematical Programming Glossary. `https://glossary.informs.org/ver2/mpgwiki/index.php?title=Cumulative_constraint`

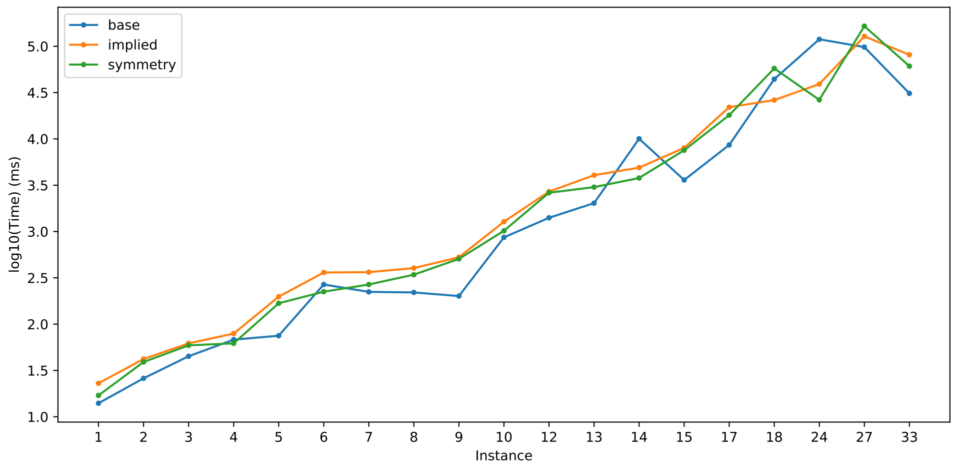
[5] Which statistics indicate an efficient run of Z3? (Stack Overflow thread). `https://stackoverflow.com/a/6847467`

Instance	Model	Time	Conflicts
1	Base	14ms	61
	Implied	23ms	41
	Symmetry	17ms	12
2	Base	26ms	138
	Implied	42ms	74
	Symmetry	39ms	97
3	Base	45ms	518
	Implied	62ms	163
	Symmetry	59ms	152
4	Base	68ms	774
	Implied	79ms	228
	Symmetry	62ms	170
5	Base	75ms	883
	Implied	198ms	617
	Symmetry	168ms	670
6	Base	268ms	3533
	Implied	361ms	1214
	Symmetry	224ms	786
7	Base	223ms	3197
	Implied	364ms	1247
	Symmetry	268ms	1224
8	Base	220ms	1691
	Implied	403ms	1154
	Symmetry	342ms	834
9	Base	201ms	1882
	Implied	527ms	2221
	Symmetry	506ms	1784
10	Base	865ms	7474
	Implied	1s 276ms	3209
	Symmetry	1s 18ms	2171
11	Implied	1m 13s	90859
	Symmetry	24s 38ms	40918
12	Base	1s 407ms	9392
	Implied	2s 698ms	5033
	Symmetry	2s 621ms	5541

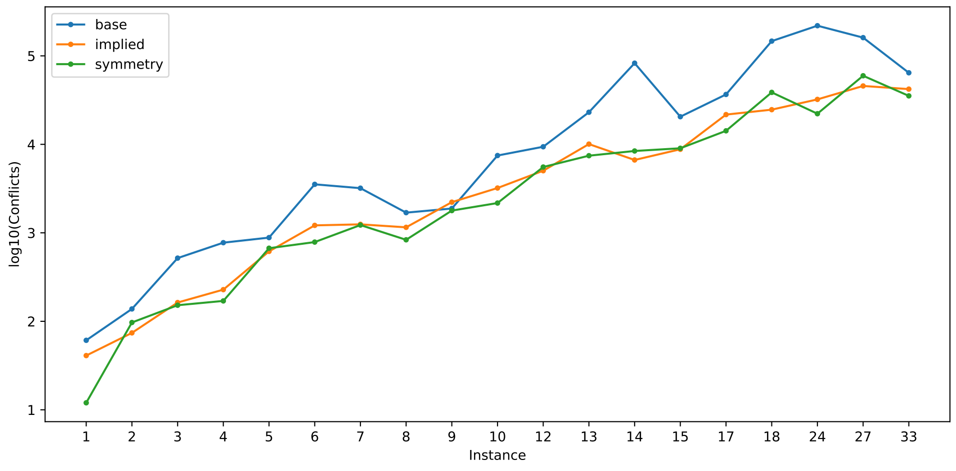
Table 3: Solving statistics of the three increasingly constrained fixed-orientation models when solved under 5 minutes by Z3. Models are shown below the second column only when able to solve the instance within the time limit. The last column refers to the `conflicts` Z3 statistic, which is roughly equal to the number of branches explored by the solver during the search process: a lower number of conflicts is therefore an indication of a more efficient axiomatization [5]. [continues in table 4]

Instance	Model	Time	Conflicts
13	Base	2s 25ms	23056
	Implied	4s 58ms	10066
	Symmetry	3s 15ms	7443
14	Base	10s 42ms	82682
	Implied	4s 894ms	6658
	Symmetry	3s 782ms	8415
15	Base	3s 598ms	20532
	Implied	7s 993ms	8796
	Symmetry	7s 542ms	9028
17	Base	8s 605ms	36635
	Implied	22s 12ms	21688
	Symmetry	18s 47ms	14219
18	Base	44s 164ms	146752
	Implied	26s 255ms	24656
	Symmetry	57s 712ms	38666
20	Implied	2m 26s	67029
	Symmetry	1m 16s	37671
21	Symmetry	4m 16s	100272
23	Implied	41s 900ms	27108
	Symmetry	1m 46s	60593
24	Base	1m 59s	219186
	Implied	39s 73ms	32202
	Symmetry	26s 390ms	22194
26	Base	4m 6s	338250
27	Base	1m 38s	160711
	Implied	2m 8s	45689
	Symmetry	2m 45s	59598
28	Implied	3m 33s	98823
	Symmetry	2m 51s	60364
31	Implied	40s 612ms	25839
	Symmetry	21s 175ms	12613
33	Base	31s 8ms	64389
	Implied	1m 21s	42101
	Symmetry	1m 1s	35299

Table 4: [continues from table 3] Solving statistics of the three increasingly constrained fixed-orientation models when solved under 5 minutes by Z3. Models are shown below the second column only when able to solve the instance within the time limit. The last column refers to the `conflicts` Z3 statistic, which is roughly equal to the number of branches explored by the solver during the search process: a lower number of conflicts is therefore an indication of a more efficient axiomatization [5].



(a) Time

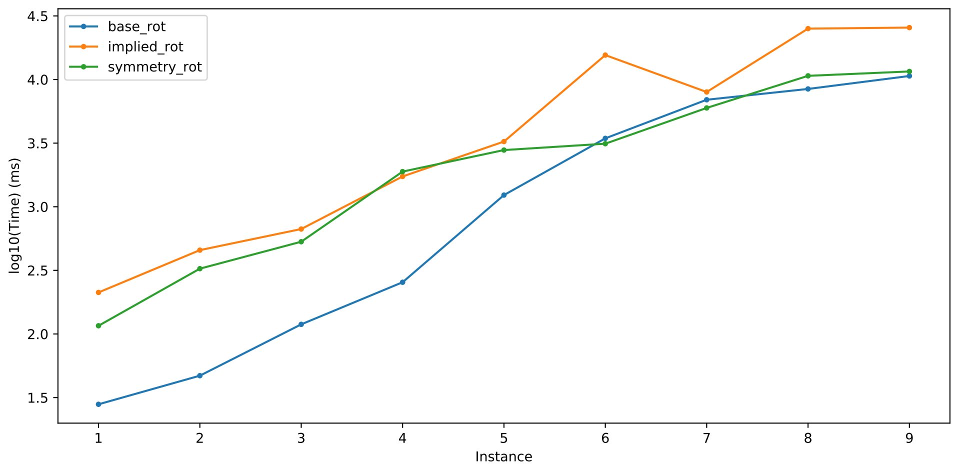


(b) Conflicts

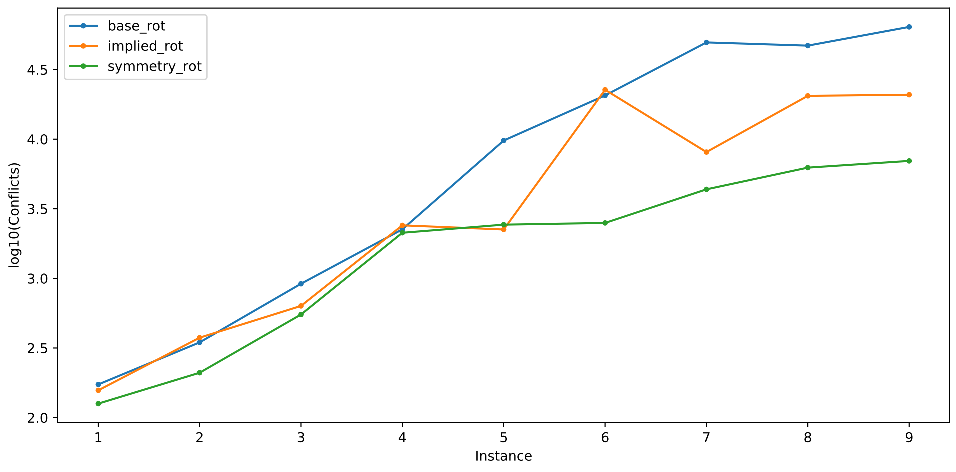
Figure 1: Comparison between Z3 solving statistics of models Base, Implied, and Symmetry on instances solved under 5 minutes by all three of them. Values on the y-axis are expressed in $\log_{10}$ scale.

Instance	Model	Time	Conflicts
1	Base-Rot	28ms	173
	Implied-Rot	212ms	157
	Symmetry-Rot	116ms	126
2	Base-Rot	47ms	347
	Implied-Rot	456ms	375
	Symmetry-Rot	326ms	210
3	Base-Rot	119ms	915
	Implied-Rot	668ms	634
	Symmetry-Rot	531ms	550
4	Base-Rot	255ms	2251
	Implied-Rot	1s 730ms	2404
	Symmetry-Rot	1s 889ms	2127
5	Base-Rot	1s 235ms	9777
	Implied-Rot	3s 254ms	2245
	Symmetry-Rot	2s 789ms	2432
6	Base-Rot	3s 448ms	20605
	Implied-Rot	15s 561ms	22632
	Symmetry-Rot	3s 130ms	2502
7	Base-Rot	6s 935ms	49537
	Implied-Rot	7s 979ms	8080
	Symmetry-Rot	5s 979ms	4361
8	Base-Rot	8s 431ms	46978
	Implied-Rot	25s 128ms	20472
	Symmetry-Rot	10s 690ms	6245
9	Base-Rot	10s 661ms	64007
	Implied-Rot	25s 593ms	20863
	Symmetry-Rot	11s 570ms	6978
10	Symmetry-Rot	3m 40s	67639

Table 5: Solving statistics of the three increasingly constrained rotation models when solved under 5 minutes by Z3. Models are shown below the second column only when able to solve the instance within the time limit. The last column refers to the `conflicts` Z3 statistic, which is roughly equal to the number of branches explored by the solver during the search process: a lower number of conflicts is therefore an indication of a more efficient axiomatization [5].

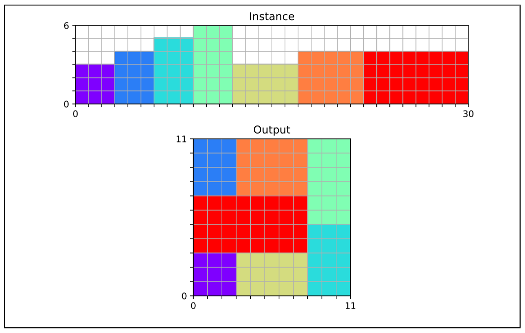


(a) Time

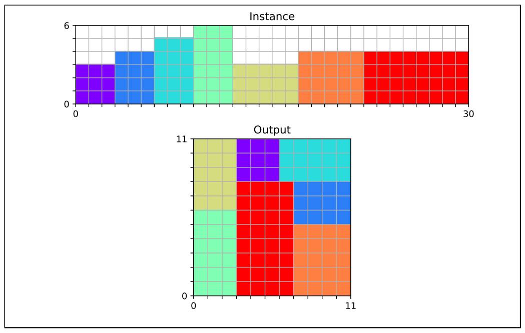


(b) Conflicts

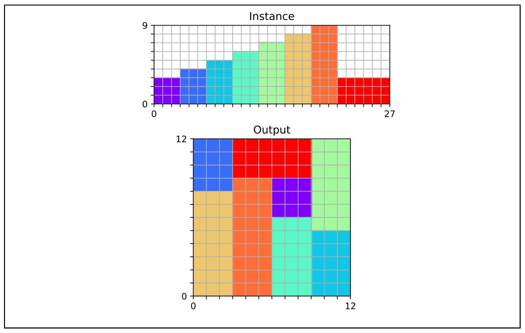
Figure 2: Comparison between Z3 solving statistics of models Base-Rot, Implied-Rot, and Symmetry-Rot on instances solved under 5 minutes by all three of them. Values on the y-axis are expressed in $\log_{10}$ scale.



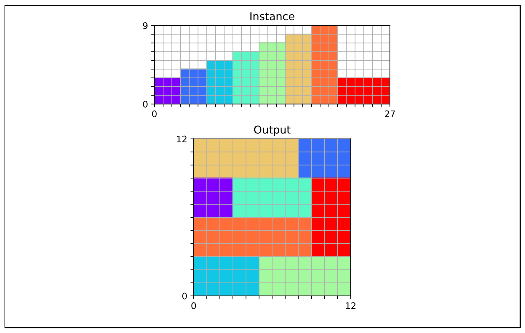
(a) Symmetry



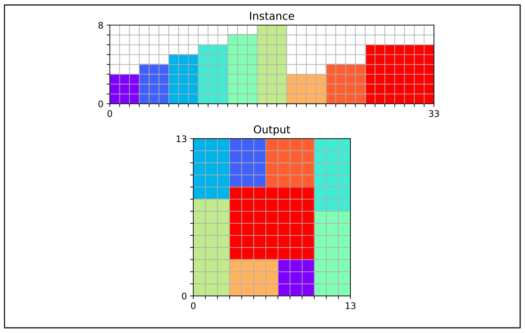
(b) Symmetry-Rot



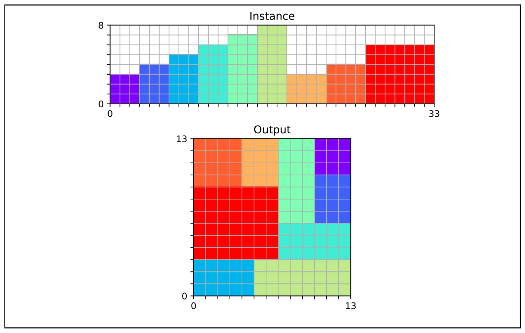
(c) Symmetry



(d) Symmetry-Rot



(e) Symmetry



(f) Symmetry-Rot

Figure 3: Solutions produced by Symmetry and Symmetry-Rot on Instance 4 (a, b), Instance 5 (c, d), and Instance 6 (e, f).