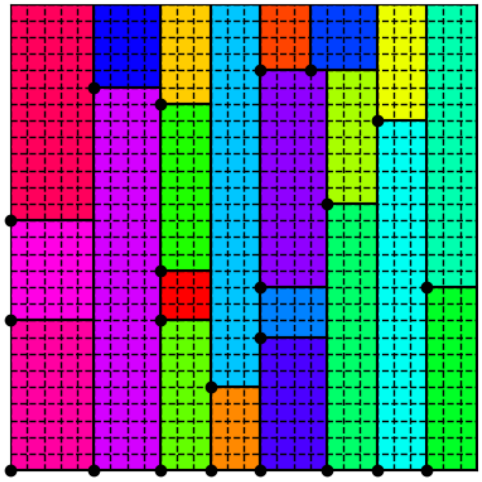
Alma Mater Studiorum - Università di Bologna

Satisfiability Modulo Theory approach to VLSI



Project for the course Combinatorial Decision Making and Optimization (module 1)

Martina Rossini
martina.rossini3@studio.unibo.it

July 2021

Contents

1 Introduction

VLSI is the problem of placing a finite number of fixed-size circuits onto a silicon plate of fixed width, while minimizing its height. This problem is very relevant nowadays, since it allows smartphones and other electronics to be extremely powerful tools while keeping them of a manageable size.

We will try to solve this problem using Satisfiability Modulo Theory (SMT), which is basically an extension of SAT that allows us to specify our problem in first-order logic instead of propositional logic and also allows us to use some domain specific reasoning. With respect to SAT, SMT improves readability, expressiveness and scalability, while causing an acceptably small loss of efficiency.

2 Initial simple model

Similarly to what we did when solving the model in Constraint Programming, we decided to keep two variables x_i and y_i for the x and y coordinates of the bottom-left corner of every rectangle $i = 1 \cdots N$, with N being the number of total circuits to place on the board. Let now h_i and w_i be the fixed height and width for rectangle $i = 1 \cdots N$, our first model simply encoded the fact that no circuit should be placed outside the boundaries of the silicon board and that circuits should not overlap. Using a first-order logic notation, and denoting as W and H, respectively, the plate's width and height, the first constraint can be written as:

$$\begin{array}{ccccc} x_i \geq 0 & \wedge & x_i + w_i \leq W & \wedge \\ y_i \geq 0 & \wedge & y_i + h_i \leq H & \forall\ i \end{array} \tag{1}$$

While the non-overlapping constraint is written like this:

$$\begin{array}{ccccc} x_i + w_i \leq x_j & \vee & x_i \geq x_j + w_j & \vee \\ y_i + h_i \leq y_j & \vee & y_i \geq y_j + h_j & \\ \forall\ i, j & \text{s.t.} & i \neq j & \end{array} \tag{2}$$

3 Refining the model

Once our initial rough model was built, we started trying to improve it: in particular we implemented the two constraints suggested in the problem

specification and we added some symmetry breaking constraints. Finally, we also tried to support rectangle rotation, even if the original problem specification did not admit it. The following subsections explore in a bit more detail how we implemented these refinements.

3.1 Implied constraints

As suggested in the problem specification, if we draw an horizontal line in y_thr and consider the set of all the intersected circuits (let's call it S), then $\sum_{j \in S} w_j \leq W$. A similar property also holds if we draw a vertical line in x_thr.

Supposing to have at our disposal two first-order functions, $If(condition, expr1, expr2)$ – which checks the truth value of *condition* and, if it holds, returns $expr1$, otherwise returns $expr2$ – and $Sum(array)$ which returns the sum of the elements inside $array$, we can encode these two constraints as:

$$\begin{aligned} Sum([If(x_i \leq x\_thr \ \wedge \ x_i + w_i > x\_thr, \ h_i, \ 0) \forall \ i = 1 \cdots N]) \leq H \\ \forall w\_thr = 0 \cdots W - 1 \end{aligned} \tag{3}$$

$$\begin{aligned} Sum([If(y_i \leq y\_thr \ \wedge \ y_i + h_i > y\_thr, \ w_i, \ 0) \forall \ i = 1 \cdots N]) \leq W \\ \forall y\_thr = 0 \cdots H - 1 \end{aligned} \tag{4}$$

Note that we did not have to define the functions Sum and If ourselves, as the SMT solver we used (the Python API to Z3, called Z3Py) already provided them for us.

3.2 Symmetry breaking constraints

The more obvious symmetries we could spot in this problem were related to the flipping of the x and y axes: we can break them quite easily by imposing a lexicographic order on the values of the rectangles x and y coordinates. To write these two symmetry breaking constraints in in FOL we first need to define two functions: $flip(v, l, max\_val)$ and $lex\_lesseq(a1, a2)$, with v, l, $a1$ and $a2$ arrays of decision variables.

$$\begin{aligned} flip(v, \ l, \ max\_val) = \\ [max_v al - v[i] - l[i] \ \forall \ i] \end{aligned} \tag{5}$$

$lex\_lesseq(a1, a2)$ simply compares the elements of a_1 and a_2 and returns true if and only if the first array comes lexicographically before the second (i.e.: only if $a1_1 \leq a2_1 \ \vee \ (a1_1 == a2_1 \ \wedge \ a1_2 \leq a2_2) \ \vee \ \dots$). Using these two functions, the symmetry breaking constraints in question can be written as:

$$\begin{aligned} &lex\_lesseq(x, flip(x, w, W)) \\ &lex\_lesseq(y, flip(y, h, H)) \end{aligned} \tag{6}$$

with $x = [x_1, x_2, \cdots, x_N]$, $y = [y_1, y_2, \cdots, y_N]$, $w = [w_1, w_2, \cdots, w_N]$ and $h = [h_1, h_2, \cdots, h_N]$.

We also considered breaking the symmetries related to the swapping of same-size rectangles but they required at least two iterations over the circuits and – upon writing the relative constraint and trying the execution on a couple of instances – we found that they significantly slowed down the search, thus not bringing us any advantage.

3.3 Supporting rectangle rotation

Finally, we supported rectangle rotation by defining the decision variables $actual\_w$ and $actual\_h$ like this:

$$\begin{aligned} (actual\_w[i] = w_i \quad &\wedge \quad actual\_h[i] = h_i) \quad \vee \\ (actual\_w[i] = h_i \quad &\wedge \quad actual\_h[i] = w_i \quad \forall \ i \end{aligned} \tag{7}$$

Note that we also considered the possibility of simply duplicating the rectangles, introducing variables x_rot_i and y_rot_i for every circuit i and imposing that the width of the rotated rectangle be h_i and its height be w_i and that – in every solution – either x_rot_i/y_rot_i or x_i/y_i can be assigned but not both at the same time. However, this second solution seemed more complex to implement and would also introduce a large number of other decision variables, which could slow down the search, particularly for large instances.

4 Experimental results

Note that we explored two different approaches to solve this problem using the SMT solver Z3 with the Python API: we first simply ran Z3's optimizer on our model and observed the results (which can be seen in Table 1). However,

since this approach seemed quite slow in producing results for our instances, we first tried to improve the model by adding other constraints (for examples, those that prohibit swapping of same-size objects) and – when we found little to no improvement – we tried a different approach. We reduced our optimization problem to a series of satisfaction problems: using the lower and upper bounds that we were easily able to compute for the height we iteratively selected the smallest height value possible and solved our problem as a satisfaction problem with that value as H. If one satisfaction problem turns out to be unsatisfiable, we simply increase the value of H by one and try again. The results obtained with this approach for the model with and without rotation can be found in Tables 2 and 3. Note that, in these tables, we only show the timing information related to the instances we were able to solve.

We can see that, although not as good as the Constraint Programming model, our SMT model is able to solve 31 out of 40 instances when we do not allow rotation and 20 instance out of 40 when we do allow our circuits to rotate. For future reference, it may be interesting to also try on these instances a model without the implied constraints suggested in the project specification as they may be significantly slowing down the computation – indeed, they require an iteration over all the circuits for every possible width and height thresholds x_thr and y_thr.

Table 1: Final results for optimization model without rotation

Instance name	**Optimal solution**
out-1.txt	0.07
out-1.txt	0.07
out-5.txt	0.30
out-1.txt	0.06
out-1.txt	0.07
out-10.txt	0.93
out-11.txt	6.05
out-12.txt	2.93
out-13.txt	2.49
out-14.txt	13.20
out-15.txt	12.57
out-16.txt	236.70
out-17.txt	84.76
out-2.txt	0.26
out-24.txt	212.43
out-3.txt	0.15
out-31.txt	190.29
out-4.txt	0.17
out-5.txt	0.39
out-6.txt	0.40
out-7.txt	0.45
out-8.txt	0.63
out-9.txt	0.78

Table 2: Final results for iterative model without rotation

Instance name	**Optimal solution**
out-1.txt	0.06
out-10.txt	0.39
out-11.txt	16.88
out-12.txt	1.92
out-13.txt	0.55
out-14.txt	1.81
out-15.txt	0.90
out-16.txt	20.29
out-17.txt	2.44
out-18.txt	13.44
out-19.txt	32.94
out-2.txt	0.08
out-20.txt	8.29
out-21.txt	49.57
out-23.txt	7.00
out-24.txt	3.20
out-26.txt	22.79
out-27.txt	12.76
out-28.txt	43.48
out-29.txt	42.56
out-3.txt	0.10
out-31.txt	10.83
out-33.txt	2.03
out-35.txt	54.67
out-37.txt	124.99
out-4.txt	0.13
out-5.txt	0.18
out-6.txt	0.23
out-7.txt	0.25
out-8.txt	0.31
out-9.txt	0.26

Table 3: Final results for iterative model with rotation

Instance name	**Optimal solution**
out-1.txt	0.06
out-10.txt	0.57
out-11.txt	58.71
out-12.txt	2.87
out-13.txt	26.34
out-14.txt	15.03
out-15.txt	76.07
out-2.txt	0.09
out-24.txt	99.23
out-3.txt	0.17
out-31.txt	101.13
out-33.txt	113.89
out-35.txt	151.47
out-36.txt	4.09
out-4.txt	0.25
out-5.txt	0.18
out-6.txt	0.69
out-7.txt	0.60
out-8.txt	0.38
out-9.txt	0.52