VLSI: VERY LARGE SCALE INTEGRATION

CDMO REPORT - BOOLEAN SATISFIABILITY PROBLEM

Lorenzo Pratesi
lorenzo.pratesi2@studio.unibo.it

Artificial Intelligence

University of Bologna

June 2022

Contents

1 Introduction

VLSI (Very Large Scale Integration) refers to the problem of integrating circuits into silicon chips. Given a fixed-width plate and a list of rectangular circuits, decide how to place them on the plate so that the length of the final plate is minimized. We will address two variants of the problem; in the first each circuit must be placed in a fixed orientation with respect to the others, while in the second case each circuit could be also rotated by 90° degrees. The solutions provided in this work are implemented in Z3Py, a Python API for Z3 (1). Z3 is a state-of-the-art SMT theorem prover; in this work only its SAT solver has been used. In Section 2 we will discuss how the problem is formally defined by means of its inputs and how the outputs are structured. in Section 3 we will see how the models are defined together with their constraints and some SAT-encoding. In Section 4 it is described the used optimization strategy. In Section 5 we will see the all the experimental setups and their results over the provided instances.

2 Problem Description

The input of the problem is structured as follows:

- w: width of the plate;
- n: number of circuits;
- m: a $n \times 2$ matrix where m_{i1} and m_{i2} are the width and the height of the i-th circuit respectively.

The aim of the problem is to fill the plate with all the circuits by minimizing the total height without exceeding its width w.

The instances of the problem are structured in the following way:

w
n
m_{11} m_{12}
m_{21} m_{22}
...
m_{n1} m_{n2}

The outputs of the problem will be structured in the following way:

w h
n
x_1 y_1 m_{11} m_{12}
x_2 y_2 m_{21} m_{22}
...
x_n y_n m_{n1} m_{n2}

where x_i and y_i are the coordinates of the bottom-left corner of the i-th circuit.

Note that, if rotation of circuits is allowed, the dimension of a circuit could be inverted.

3 Modelling

We are now going to introduce the models and the constraints used to address the VLSI problem. We are going to represent each circuit placed in the plate by its bottom-left coordinate. Two types of models have been defined:

- **Standard**: the model that uses all the constraints described below, without allowing rotation of circuits;

- **Rotation**: like the Standard model, but allowing rotation.

Both types of models take as input all the inputs defined in the previous section, but also a constant h representing the maximum height for the current model. This choice is motivated by the optimization strategy (see Section 4)

3.1 Variables

For each circuit i, we are going to define a set of Boolean variable for each axis of the plate. For each circuit, along the width axis, w new Boolean variables have been defined, while along the height axis h new variables. From now on, we will refer to those variables as "axis variables". Assigning true to a variable k means that the bottom-left coordinate along that axis is k. More formally we define:

- x_{ik} variables with $i \in \{1, ..., n\}, k \in \{0, ..., w-1\}$ for the width axis;
- y_{ik} variables with $i \in \{1, ..., n\}, k \in \{0, ..., h-1\}$ for the height axis.

If rotation is allowed, for each circuit i a Boolean variable r_i is defined, representing the circuit orientation.

3.2 Constraints

3.2.1 Exactly one constraints

For each circuit, we must impose that there exists exactly one axis variable true for each axis. As we know, given a set of literals X, the exactly one constraint is obtained as:

$$exactly\_one(X) \equiv at\_least\_one(X) \wedge at\_most\_one(X)$$

where $at\_least\_one(X) \equiv \bigvee_{x \in X} x$ and $at\_most\_one(X)$ could be represented using different encodings (see Section 3.3)

For this particular problem, two "exactly one" constraints have been defined:

$$exactly\_one\_x \equiv \bigwedge_{1 \leq i \leq n} exactly\_one(\{x_{i0}, ..., x_{iw-1}\})$$

$$exactly\_one\_y \equiv \bigwedge_{1 \leq i \leq n} exactly\_one(\{y_{i0}, ..., y_{ih-1}\})$$

3.2.2 No over w/h constraint

We must ensure that all the circuits do not exceed the plate both horizontally and vertically. The following constraint ensure this behavior:

$$no\_over\_w \equiv \bigwedge_{i=1}^{n} \bigwedge_{j=w-m_{i1}+1}^{w} \neg x_{ij}$$

$$no\_over\_h \equiv \bigwedge_{i=1}^{n} \bigwedge_{j=h-m_{i2}+1}^{h} \neg y_{ij}$$

If rotation is allowed:

$$no\_over\_w \equiv \bigwedge_{i=1}^{n} (\neg r_i \implies \bigwedge_{j=w-m_{i1}+1}^{w} \neg x_{ij}) \wedge (r_i \implies \bigwedge_{j=h-m_{i2}+1}^{h} \neg x_{ij})$$

$$no\_over\_h \equiv \bigwedge_{i=1}^{n} (\neg r_i \implies \bigwedge_{j=h-m_{i2}+1}^{h} \neg y_{ij}) \wedge (r_i \implies \bigwedge_{j=w-m_{i1}+1}^{w} \neg y_{ij})$$

3.2.3 No overlap constraint

In order to ensure that each circuit will not be placed one on top of the other (so to not overlap) for each pair of circuits (i, j) we must be sure that, if $k1$ represents one of its bottom-left coordinates, then $(x_{ik1} \implies \bigvee_{k2=k1+m_{i1}}^{w} x_{jk2}) \vee (y_{ik1} \implies \bigvee_{k2=k1+m_{i2}}^{h} y_{jk2})$. If one of those conditions holds, it means

that one circuit is placed in front of the other along the considered axis, so they do not overlap. The following constraint ensure this behavior:

$$no\_overlap \equiv \bigwedge_{i=1}^{n} \bigwedge_{j=i+1}^{n}$$
$$[\bigwedge_{k1=1}^{w-m_{i1}+1} (x_{ik1} \implies \bigvee_{k2=k1+m_{i1}}^{w} x_{jk2}) \vee \bigwedge_{k1=1}^{w-m_{i1}+1} (x_{jk1} \implies \bigvee_{k2=k1+m_{j1}}^{w} x_{ik2}) \vee$$
$$\bigwedge_{k1=1}^{h-m_{i2}+1} (y_{ik1} \implies \bigvee_{k2=k1+m_{i2}}^{h} y_{jk2}) \vee \bigwedge_{k1=1}^{h-m_{i2}+1} (y_{jk1} \implies \bigvee_{k2=k1+m_{j2}}^{h} y_{ik2})]$$

If rotation is allowed:

$$no\_overlap \equiv \bigwedge_{i=1}^{n} \bigwedge_{j=i+1}^{n}$$
$$[(\bigwedge_{k1=1}^{w-m_{i1}+1} (x_{ik1} \wedge \neg r_i) \implies \bigvee_{k2=k1+m_{i1}}^{w} x_{jk2}) \wedge (\bigwedge_{k1=1}^{w-m_{i2}+1} (x_{ik1} \wedge r_i) \implies \bigvee_{k2=k1+m_{i2}}^{w} x_{jk2}) \vee$$
$$(\bigwedge_{k1=1}^{w-m_{i1}+1} (x_{jk1} \wedge \neg r_j) \implies \bigvee_{k2=k1+m_{j1}}^{w} x_{ik2}) \wedge (\bigwedge_{k1=1}^{w-m_{i2}+1} (x_{jk1} \wedge r_j) \implies \bigvee_{k2=k1+m_{j2}}^{w} x_{ik2}) \vee$$
$$(\bigwedge_{k1=1}^{h-m_{i2}+1} (y_{ik1} \wedge \neg r_i) \implies \bigvee_{k2=k1+m_{i2}}^{h} y_{jk2}) \wedge (\bigwedge_{k1=1}^{h-m_{i1}+1} (y_{ik1} \wedge r_i) \implies \bigvee_{k2=k1+m_{i1}}^{h} y_{jk2}) \vee$$
$$(\bigwedge_{k1=1}^{h-m_{i2}+1} (y_{jk1} \wedge \neg r_j) \implies \bigvee_{k2=k1+m_{j2}}^{h} y_{ik2}) \wedge (\bigwedge_{k1=1}^{h-m_{i1}+1} (y_{jk1} \wedge r_j) \implies \bigvee_{k2=k1+m_{j1}}^{h} y_{ik2})]$$

3.3 Encodings

In this work, four different encodings of the at_most_one constraint have been defined and implemented. We are going how they are defined and what are their strength and weaknesses. We define X as a set of Boolean variables and $n = |X|$.

3.3.1 Pairwise encoding

The pairwise encoding simply ensure that in every pair of Boolean variables, just one can hold or none.

$$at\_most\_one\_pw(X) \equiv \bigwedge_{1 \leq i \leq n} \bigwedge_{i+1 \leq j \leq n} \neg(x_i \wedge x_j)$$

This encoding does not introduce any additional variable, but the number of clauses is in the order of $O(n^2)$.

3.3.2 Sequential encoding

The sequential encoding uses $n-1$ new variables s_i to keep track of which x_i if true. Each s_i if true indicates that the sum has reached 1 by i, in this way we can say that, if x_i is true then also s_i must be true and if s_j is true then $x_j + 1$ must be false, ensuring that only one $x \in X$ must be true. The following is the definition of the encoding:

$$at\_most\_one\_pw(X) \equiv (\neg x_1 \vee s_1) \wedge (\neg x_n \vee \neg s_{n-1}) \wedge (\bigwedge_{1 < i \leq n-1} ((\neg x_i \vee s_i) \wedge (\neg s_{i-1} \vee s_i) \wedge (\neg x_i \vee \neg s_{i-1})))$$

This encoding introduces $n-1$ new variables and $3n-4$ $(O(n))$ new clauses.

3.3.3 Bitwise encoding

The bitwise encoding introduces m new variables r_j where $m = log_2\, n$. Then it imposes that, if x_i is true, then every bit of the binary encoding of i, represented by r_j or its negation, must hold. In this we we assure index uniqueness and so at most one variable can hold. The following constraint ensure this behavior.

$$at\_most\_one\_bw(X) \equiv \bigwedge_{1 \leq i \leq n} \bigwedge_{1 \leq j \leq m} \neg x_i \wedge r_{ij}[\neg r_{ij}]$$

where r_{ij} if bit j of the binary encoding of $i-1$ is 1, else $\neg r_{ij}$

This encoding introduces m new variables and $nlog_2(n)$ clauses

3.3.4 Heule encoding

The Heule encoding uses a recursive definition to define the constraint. It is defined as:

$$at\_most\_one\_he(X) \equiv \begin{cases} at\_most\_one\_pw(X) & if \;\; n \leq 4 \\ at\_most\_one\_pw(\{x_1, x_2, x_3, y\}) \wedge at\_most\_one\_he(\{x_4, ..., x_n, \neg y\}) \end{cases}$$

Basically, if $n \leq 4$ it applies the pairwise encoding, otherwise, it introduces a new variable y and defines the above constraint; if y holds then no variables in the first half must hold, while in the second half at most one variable must hold, by calling again the same constraint recursively.

It introduces $(n-3)/2$ new variables and $3n-6$, $O(n)$ new clauses.

4 Optimization strategy

In order to find an optimal solution to the problem, so to minimize the height h, the strategy used is to redefine the model with an increased height every time the solver founds an unsatisfability. First, three constants have been defined:

- $area_{min} = \sum_{i=1}^{n} m_{i1} m_{i2}$
- $min_h = max(max(m_{02}, ..., m_{n2}), \lfloor area_{min}/w \rfloor)$
- $max_h = \sum_{i=1}^{n} m_{i2}$

If rotation is allowed:

- $area_{min} = \sum_{i=1}^{n} m_{i1} m_{i2}$
- $min_h = max(max(min(m_{01}, m_{02}), ..., min(m_{n1}, m_{n2})), \lfloor area_{min}/w \rfloor)$
- $max_h = \sum_{i=1}^{n} max(m_{i1}, m_{i2})$

where $area_{min}$ represents the minimum area of the plate covered by the circuits, min_h represents the minimum height possible and max_h represents the maximum height possible.

Starting with $h = min_h$, we define the model according to that height, then if it is satisfiable we found and optimal solution, otherwise redefine the model with height equal to $h + 1$ and repeat the process until satisfability or maximum height (max_h) reached.

5 Experimental setup and Results

In this section there will be showed the experimental setup and the results obtained on the instances provided. For each type of model (standard and rotation allowed) it was interesting to see how it would perform by changing the $exactly\_one$ encoding, so every type of encoding has been experimented. Table 1 shows some solving statistics for each implemented model. The results obtained showed that the best encoding for the standard model was the pairwise one, while for the rotation model was the sequential one. Probably the problem implemented in this way prefers a higher number of clauses rather than a higher number of variables. Figure 1 shows the solving times of the best models of each type of model over all instances. As expected, the higher number of variables of the rotation model affect the solving speed.

Table 1: Timing statistics with different encodings

Model	Endcoding	Solved instances	Max solving time	Avg solving time
Standard	Pairwise	**33**	260.6666298828125	34.4474
Standard	Sequential	31	238.66585717773438	35.9456
Standard	Bitwise	32	210.66489599609375	35.384
Standard	Heule	30	110.81884960937501	17.7101
Rotation	Pairwise	20	275.1914113769531	23.3959
Rotation	Sequential	**22**	287.6093642578125	80.9692
Rotation	Bitwise	19	294.6655278320313	46.5
Rotation	Heule	17	78.01636840820312	10.2177

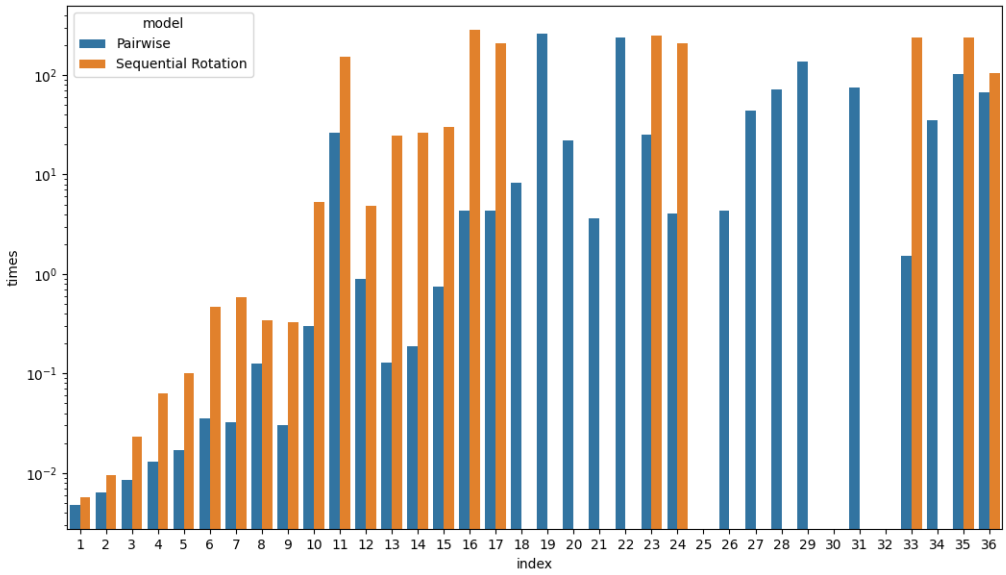

Figure 1: Solving times of the best models over all the instances

References

[1] Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *TACAS*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008.