VLSI: VERY LARGE SCALE INTEGRATION

CDMO REPORT - SATISFIABILITY MODULO THEORIES

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 of the Z3 solver. Z3Py provides an interface to define SMT models in Python but natively supports only Z3 as solver. In order to be able to use other solvers while using the Z3Py's model definition, it has been used the `to_smt2()` method of its `Solver` class. In this way all the constraints have been implemented in Z3Py while allowing to solve the models with other solvers.

Three different solvers have been used in this work:

- **Z3**: a state-of-the-art theorem prover developed by Microsoft (4);
- **CVC4**: an open-source theorem prover for SMT problems (3);
- **CVC5**: the successor of CVC4 (2).

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. In Section 4 there are described all the optimization strategies used to solve the problem. 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 with and SMT approach. 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.

All the models are implemented following the SMT-LIB Quantifier-Free Integer Difference Logic `QF\_IDL`. This theory impose to have formulas with atoms of the form $x - y \bowtie k$ where x, y are integer variables, k is an integer constant and $\bowtie \in \{<, \leq, >, \geq, =, \neq\}$ (1). By constraining some types of operations, so using a specified logic, the solver may be able to apply some specialized and more efficient techniques.

In order to create tighter bounds, some useful 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.

3.1 Variables

For each circuit i, we are going to define two sets of integer variables. For the width axis, $x_i \;\; i \in \{1, ..., n\}$ variables have been defined, while for the height axis, $y_i \;\; i \in \{1, ..., n\}$ variables have been defined.

The maximum height covered by all the positioned circuits has been defined as an integer variable h. Our aim is to minimize that variable.

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 Rotation model constraints

Some constraints have been defined in a different way when rotation is allowed, but one it is implemented only for that particular model. Some solvers allows to not assign a truth value to variables if their value do not affect the final result. It is the case of circuits having the same size, so $m \times m$ circuits. In order to ensure that a solver will assign a truth value to the r_i corresponding to the circuit having equal sizes, the following constraint has been defined; suppose R is the set of indexes of the circuits having the same size:

$$rot\_fix \equiv \bigwedge_{i \in R} r_i$$

In this way we assign true to those circuits.

The rotation model requires also to invert sizes if the model is rotated. The following formulas ensure to use the right size for a particular circuit and are defined following `QF_IDF`:

$$check\_inequality\_dx(i, v) \equiv (\neg r_i \implies x_i - v \leq -m_{i1}) \land (r_i \implies x_i - v \leq -m_{i2}) \equiv$$
$$(r_i \lor x_i - v \leq -m_{i1}) \land (\neg r_i \lor x_i - v \leq -m_{i2})$$

$$check\_inequality\_dy(i, v) \equiv (\neg r_i \implies y_i - v \leq -m_{i2}) \land (r_i \implies y_i - v \leq -m_{i1}) \equiv$$
$$(r_i \lor y_i - v \leq -m_{i2}) \land (\neg r_i \lor y_i - v \leq -m_{i1})$$

where v is a generic integer and i is the index of a circuit. This formulas will replace inequalities where the size of a circuit matters.

3.2.2 Bounds constraints

We know that each circuit of the plate must be placed in non-negative coordinates and its position plus its size can not exceed the maximum plate size along an axis. The following constraints implement the mentioned behavior:

- $x\_bounds \equiv \bigwedge_{1 \leq i \leq n} x_i \geq 0 \land x_i \leq w - m_{i1}$
- $y\_bounds \equiv \bigwedge_{1 \leq i \leq n} y_i \geq 0 \land y_i \leq h - m_{i2}$

If rotation is allowed:

- $x\_bounds \equiv \bigwedge_{1 \leq i \leq n} x_i \geq 0 \land check\_inequality\_dx(w, i)$
- $y\_bounds \equiv \bigwedge_{1 \leq i \leq n} y_i \geq 0 \land check\_inequality\_dy(h, i)$

3.2.3 Height bounds

We know the lower and upper bounds of that the height of the plate can reach; in order to apply those bounds the following constraint has been defined:

$$h\_bounds \equiv min_h \leq h \land h \leq max_h$$

3.2.4 No overlap constraint

To ensure that the circuits will not overlap, for each pair of circuits i, j we must ensure that $x_i + m_{i1} \leq x_j \lor y_i + m_{i2} \leq y_j$. So if one of that inequalities holds, it means that j is placed after i along one axis, and this is sufficient for stating that the two do not overlap. The following constraint ensure that behavior:

$$no\_overlap \equiv \bigwedge_{1 \leq i \leq n} \bigwedge_{i+1 \leq j \leq n}$$
$$x_i - x_j \leq -m_{i1} \lor x_j - x_i \leq -m_{j1} \lor$$
$$y_i - y_j \leq -m_{i2} \lor y_j - y_i \leq -m_{j2}$$

If rotation is allowed:

$$no\_overlap \equiv \bigwedge_{1 \leq i \leq n} \bigwedge_{i+1 \leq j \leq n}$$
$$check\_inequality\_dx(x_j, i) \lor check\_inequality\_dx(x_i, j) \lor$$
$$check\_inequality\_dy(y_j, i) \lor check\_inequality\_dy(y_i, j)$$

3.2.5 Height bound constraint

We will see in Section 4 that all the optimization strategies require a constraint to be added during the search process. The following constraint allows to define the upper bound of the height of the plate:

$$height\_constraint \equiv h \leq b$$

where b is a integer constant.

4 Optimization strategies

As we said before, the aim of the solver is to minimize the variable h. In order to do so, four different minimization strategies have been implemented. For each of those, first we define the model in the SMT-LIB language with all the variables, constraints and bounds defined in Section 3, then at each step of the search, a new height bound constraint is added. Let's define the optimization strategies:

- **Incremental**: this strategy take advantage of the push and retract operations allowed by SMT-LIB, `(push 1)` and `(pop 1)` respectively. Starting by pushing the $h \leq min_h$ constraint, we check the satisfability of the model, if SAT then the optimal h is found, otherwise we retract the last constraint and push the constraint with a one-increased height bound and repeat until maximum height is UNSAT or SAT solution found.
- **Decremental**: this strategy take advantage of decreasing direction of the height bound constraint. Starting by imposing $h \leq max_h$, we check the satisfability of the model, if SAT then we impose a one-decreased height bound constraint and repeat until UNSAT found; this means that the previous SAT bound was the optimal one. If no SAT found then the instance is UNSAT.
- **Binary search**: this strategy take advantage of the increasingly ordered domain of the plate's height, the necessary condition to start a binary search. It acts like a normal binary search, but in order to decide in which half of the domain to search, we need to see if the current model is SAT (go in the left part) or UNSAT (go in the right part). If the current model is UNSAT, we need to retract the last constraint and then push the new one, otherwise we simply add a new tighter height bound.
- **Redefine**: this strategy simply redefines the entire model when an unsatisfability with a current height bound is found. Starting from $h \leq min_h$, we check the satisfability of the model, if SAT then the optimal h is found, otherwise we redefine the entire model with the one-increased height bound constraint. Repeat untill SAT found or max_h reached.

The Redefine strategy has been implemented as an alternative to the Incremental one because if the `QF_IDL` logic is defined, the Z3 solver is significantly slower when the `(push 1)` and `(pop 1)` operations are used. The logic definition probably affected the way Z3 solves instances with the push and retract operations.

5 Experimental setup and Results

In this section there will be showed the experimental setup and the results obtained on the instances provided. The experimental setup is featured by the use of three different solvers: Z3, CVC4 and CVC5; for each solver, all the optimization strategies have been experimented.

Table 1 shows some solving statistics obtained by using the Standard model and all the configurations described above. The best configuration, by means of solved instances, was using Z3 as solver and the Redefine optimization strategy. The best optimization strategy for CVC5 was the Incremental one, while for CVC4 the Redefine and the Decremental solved the same number of instances with similar maximum and average solving times. Note that is really likely that the Redefine strategy worked so well because all the optimal heights of all the instances coincided with min_h, the starting point of the search; it is also true that also the Incremental model has the same starting point, but as previously stated it was found that in Z3 push and retract operations had some troubles while using a pre-defined logic.

Figure 1 shows the solving times over all the instances computed by each solver with its corresponding best optimization strategy. Z3 beats all the other solvers in all the instances.

Table 2 shows a comparison between the solving statistics of the Standard and Rotation model, using Z3 and the Redefine strategy. As expected, the Standard model resulted the fastest.

Table 1: Timing statistics using different solvers and strategies

Solver	Optimization strategy	Solved instances	Max solving time	Avg solving time
Z3	Redefine	**38**	266.3540236816406	24.6175
Z3	Incremental	31	226.54987475585938	25.5367
Z3	Decremental	31	272.53239379882814	46.5508
Z3	Binary search	32	285.10383032226565	45.6657
CVC5	Redefine	29	233.56778393554688	35.9217
CVC5	Incremental	**32**	277.77684130859376	58.0543
CVC5	Decremental	31	298.85855419921876	45.9779
CVC5	Binary search	31	297.089037109375	49.3424
CVC4	Redefine	**35**	269.52463793945316	36.7653
CVC4	Incremental	33	271.4491970214844	24.5996
CVC4	Decremental	**35**	261.76064355468753	37.0275
CVC4	Binary search	33	211.45699926757814	25.9008

Table 2: Timing statistics for the both the models using Z3 and Redefine

Model	Solved instances	Max solving time	Avg solving time
Standard	**38**	266.3540236816406	24.6175
Rotation	31	226.54987475585938	25.5367

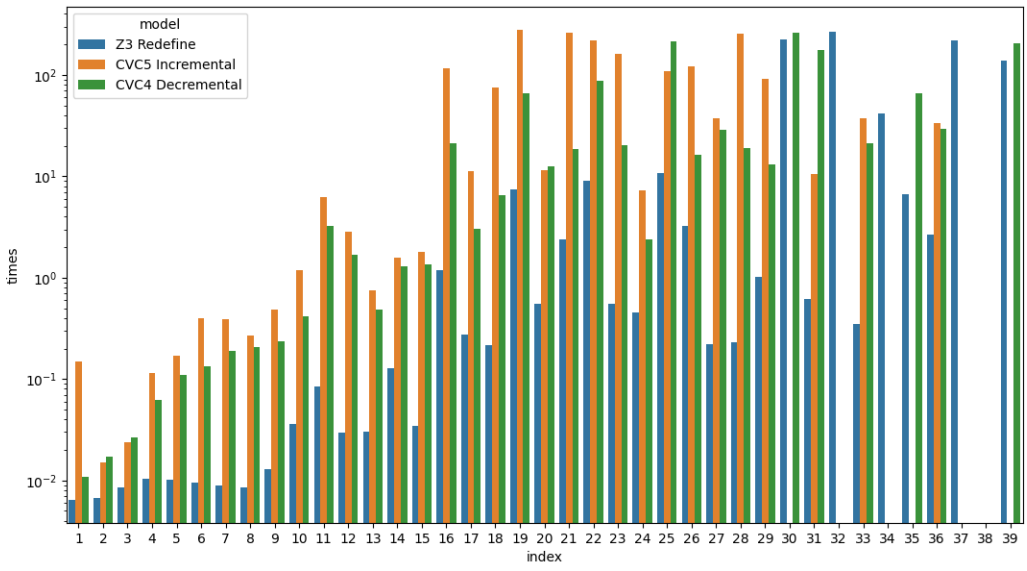

Figure 1: Solving times of the best models over all the instances

References

[1] Smt-lib the satisfiability modulo theories. https://smtlib.cs.uiowa.edu/logics-all.shtml#QF_IDL. 2022-06-21.

[2] Haniel Barbosa, Clark W. Barrett, Martin Brain, Gereon Kremer, Hanna Lachnitt, Makai Mann, Abdalrhman Mohamed, Mudathir Mohamed, Aina Niemetz, Andres Nötzli, Alex Ozdemir, Mathias Preiner, Andrew Reynolds, Ying Sheng, Cesare Tinelli, and Yoni Zohar. cvc5: A versatile and industrial-strength SMT solver. In Dana Fisman and Grigore Rosu, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part I*, volume 13243 of *Lecture Notes in Computer Science*, pages 415–442. Springer, 2022.

[3] Clark W. Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanovic, Tim King, Andrew Reynolds, and Cesare Tinelli. CVC4. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, volume 6806 of *Lecture Notes in Computer Science*, pages 171–177. Springer, 2011.

[4] 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.