Combinatorial Decision Making and Optimization

Project Report

VLSI Design

Gee Jun Hui Leonidas Yunani
Anna Fabris

Academic Year: 2020 / 2021

Contents

1 Introduction

The VLSI (Very Large Scale Integration) problem refers to the trend of integrating circuits into silicon chips. Such problems involve the development of search strategies to determine the optimal circuit placements on the silicon chips. Depending on the dimensions of the circuits and the silicon chip, the search space that must be traversed may be very large. As such, the program should be optimised to determine the circuit placements within a specified time constraint, which is the main objective of the project.

Here, SMT is used to model the problem. The model will minimize the height of the silicon chip, while determining each circuit's placement on the chip.

2 Input & Output

The inputs to be fed to the program consists of a set of 40 text files. Each text file contains the following information:

- The width of the silicon chip (`width`).
- The number of circuits (`n`).
- The width and height of each individual circuit ($W_c, H_c \ \forall c$ in $1, .., n$).

```
8 ⟵ width
4 ⟵ n
3 3 ⟵ W_1 and H_1
3 5 ⟵ W_2 and H_2
5 3 ⟵ W_3 and H_3
5 5 ⟵ W_4 and H_4
```

The program should read each text file and output the following information:

- The width and height of the silicon chip (`width`, `height`).
- The number of circuits (`n`).
- The width, height, lower left x-position and lower left y-position of each individual circuit ($W_c, H_c, Px_c, Py_c \ \forall c$ in $1, .., n$).

```
8 8 ⟵ width and height
4 ⟵ n
3 3 0 0 ⟵ W_1, H_1, Px_1 and Py_1
3 5 0 3 ⟵ W_2, H_2, Px_2 and Py_2
5 3 3 0 ⟵ W_3, H_3, Px_3 and Py_3
5 5 3 3 ⟵ W_4, H_4, Px_4 and Py_4
```

Additionally, a visualisation of the solved instance is also saved as an image in the output folder.

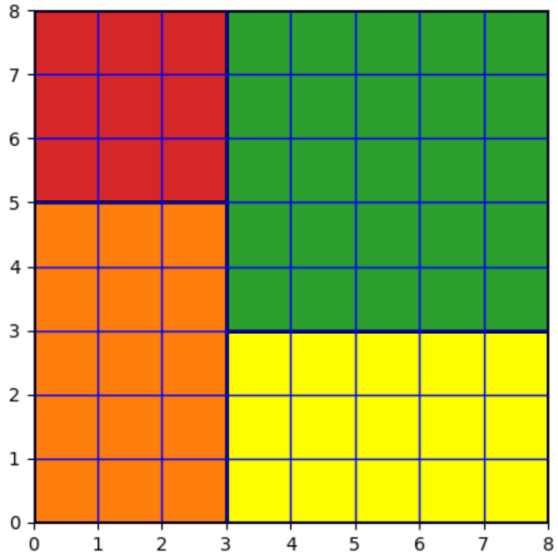


Figure 1: A visualization of a solved instance

3 Dependencies

The program has been written using the following software and libraries:

- The Matplotlib library to allow for the visualisation of the solved instances as part of the output.
- The tqdm library to allow for the tracking of the solving process for all instances.

Users can simply place the set of instances to be solved in the input folder and run the program from the command line interface.

4 SMT

4.1 Variables

Firstly, we defined some extra variable starting from the input variables:

- `circuit_widths` is an array of the *given* widths (H_c $\forall c$ in $1, .., n$) of all circuits.
- `circuit_heights` is an array of the *given* heights (H_c $\forall c$ in $1, .., n$) of all circuits.

- `max_height` is the *calculated* upper bound of the silicon chip height.

$$max\_heights = \sum_{c=1}^{n} H_c$$

- `min_height` is the *calculated* lower bound of the silicon chip height.

$$A_c = W_c * H_c$$

$$min\_height = \frac{\sum_{c=1}^{n} A_i}{max\_width}$$

Where A_i is the calculated area of the c^{th} circuit.

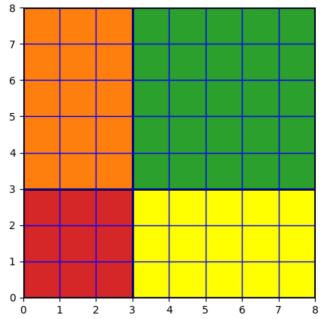
Following we specify that the output variable `height` ranges from `min_height` to `max_height`. To find the solution with the smallest `height` the SMT model will be used with increasing values of `height`.

The SMT model is defined as an array that associate each circuit `c` to its lower-left corner position `(x, y)`.

```
corners = [[Int(f"c_{i}_{pos}") for pos in range(2)] for i in
↪  range(n)]
```

c	x, y
0	0, 0
1	0, 3
2	3, 0
3	3, 3

(a) A representation of a possible internal state of `corners`



(b) A visualization of the corresponding solution

4.2 Constraints

- **Inside**

  Each circuit's bottom-left corner needs to be in the grid so that all the circuit is contained in the plate.

$\forall c$ in $1..n : Px_c \geq 0$

$\forall c$ in $1..n : Px_c + W_c \leq width$

The same two cases can be seen for the height and y coordinate of the circuits.

- **Overlap**

  Each circuit cannot overlap with another circuit. This means that the distance between the two corners needs to be at least as big as the dimension of the first circuit.

  $Con_1 : \forall c1, c2 \text{ in } 1..n \land c1 \neq c2 \land Px_{c1} \geq Px_{c2} : Px_{c1} - Px_{c2} \geq W_{c2}$

  $Con_2 : \forall c1, c2 \text{ in } 1..n \land c1 \neq c2 \land Px_{c1} < Px_{c2} : Px_{c2} - Px_{c1} \geq W_{c1}$

  The same two cases can be seen for the height and y coordinate of the circuits (Con_3 and Con_4).

  It's enough that at least one this constraints is true for the circuits to avoid overlapping.

  $Con_{overlap} : \text{Or}(Con_1, Con_2, Con_3, Con_4)$

4.3 Rotation

To allow the model the possibility of rotation we can add a list of boolean variables that is the same length as the number of circuits and keeps track of whether the circuit is rotated or not.

```
rotation = [Bool(f"r_i") for i in range(n)]
```

Given this variable, it's easy to find the actual width and height of each circuit (if the circuit is rotated width and heights need to be swapped) and using $actual\_W_c$ and $actual\_H_c$ instead of W_c and H_c in the constraints.

$$\forall c \text{ in } 1..n \; actual\_W_c = \begin{cases} H_c, & \text{if rotation[c] is true} \\ W_c, & \text{otherwise} \end{cases}$$

$$\forall c \text{ in } 1..n \; actual\_H_c = \begin{cases} W_c, & \text{if rotation[c] is true} \\ H_c, & \text{otherwise} \end{cases}$$

4.4 Final Considerations

The implied constraints seem to generally worsen the solving time (although in some specif case they do help), so they were removed from the code and the model.

The model was able to solve 37/40 instances with the time constraint of 300 seconds. Many of which can be solved in less than 60 seconds.