VLSI CP

Luca Fabri, luca.fabri@studio.unibo.it
Giuseppe Morgese, giuseppe.morgese2@studio.unibo.it

September 5, 2022

1 Introduction

Constraint Programming is a paradigm for solving combinatorial problems based on feasability rather than optimization and focuses on the constraints and variables' domain rather than the objective function. This does not mean that it can not deal with optimization problems. In fact, in this report a CP approach to solve the VLSI problem is shown.

2 Modeling

The modeling began following the advice given in the project's presentation. Thus the main variables considered are the coordinates of the bottom-left corner of each chip.

2.1 Lower and Upper bounds definition

One of the first issues addressed was deciding the variables' bounds. This step was extremely important given that it greatly reduces the search space. Regarding the chips' coordinates, the chosen ranges were:

$$x :: [0, width - min(w)] \tag{1}$$
$$y :: [0, height - min(h)] \tag{2}$$

with w and h being the array containing the chips' width and height respectively.

The bounds of the optimized variable required a more thorough work. In fact, the lower bound chosen is the maximum between the height of the tallest chip and the continuous lower bound defined as

$$L_c = \lceil \sum_j w_j h_j / W \rceil.$$

Thus the overall lower bound is

$$L_0 = max(max\_y, L_c)$$

The higher bound was computed as

$$H_0 = \left\lceil \frac{n\_circuits * max\_y}{\lfloor \frac{width}{max\_x} \rfloor} \right\rceil$$

The resulting range is:

$$h :: [L_0, H_0]$$

2.2 Basic constraints

Once the variables and their bounds were defined, the next step was to design the main constraints. The basic constraints specify that chips can't exceed the board in both directions:

$$x_i + w_i \leq width \tag{3}$$
$$y_i + h_i \leq height \tag{4}$$

Right after the no-overlap constraints were designed. To achieve such behavior, the following were implemented:

$$x_i + w_i \leq x_j \quad or \tag{5}$$
$$x_j + w_j \leq x_i \quad or \tag{6}$$
$$y_i + h_i \leq y_j \quad or \tag{7}$$
$$y_j + h_j \leq y_i \tag{8}$$

2.3 Symmetry breaking

Once the main constraints were dealt with, the natural follow-up was to break symmetries. Indeed there were different possible swaps between chips to avoid. To deal with it, firstly the biggest chip was put in the bottom-left corner.
Right after swaps between chips assuming the same x coordinate during search while having the same width were avoided. Later on, the swaps between blocks with same x assigned, different width but same height were taken into account.

2.4 Rotation

To handle rotation it had to be possible to swap w_i with h_i. To make it viable an array of boolean variables, z, was used along with the following two variables:

$$rot\_w_i = h_i * z_i + w_i(1 - z_i) \tag{9}$$
$$rot\_h_i = w_i * z_i + h_i(1 - z_i) \tag{10}$$

These variables substitute circuits' width and height in each constraint to make rotation possible.

3 Development

To implement the CP model MiniZinc was used. MiniZinc is a constraint modeling language that works in a solver-independent way and has a large library of predefined constraints.

Its wide range of built-in constraints helped a lot during development. As a matter of fact, the main constraints were implemented exploiting the analogy between VLSI and the scheduling problem. Thus to implement (1) and (2), the cumulative constraint was used.

Regarding the no-overlap constraints, the diffn constraint came in handy. In fact, it is thought to deal with 2D non-overlapping rectangles which is exactly what was needed.

To avoid symmetries, symmetry breaking constraints were implemented. Specifically, the biggest chip was placed in the bottom-left corner with the help of a sorting array, while the constraint avoiding swaps between chips with the same width or different width but same height had to be implemented by hand using if conditions.

In order to permit rotation, a separate model was used. Indeed, it was needed to substitute standard widths and heights with rotated ones. The implemented constraints directly follow (9) and (10) showed in section 2.

4 Search Strategy

One of the main perks of CP is the possibility to interact with the search process. This can be done by specifying different search and restart strategies. Clearly these strategies are not really part of the model but significantly affect the results.

MiniZinc does not use a specific strategy by standard and simply solves the problem calling the solver letting it to handle the search process. At the same time it can be specified how to carry out the search, this is done using search annotations. There are four basic search annotations:

1. int_search
2. bool_search
3. set_search
4. float_search

Given the integer domain of the problem, int_search was chosen. The main parameters of int_search are the variables, the variable choice annotation and the way to constraint a variable.

The possible variable choice annotations are:

- input_order

- first_fail
- smallest
- dom_w_deg

Because of the presence of a sorting array and by better performance, the input_order variable choice annotation was chosen.

The possible ways to constrain a variable are:

- indomain_min
- indomain_median
- indomain_random
- indomain_split

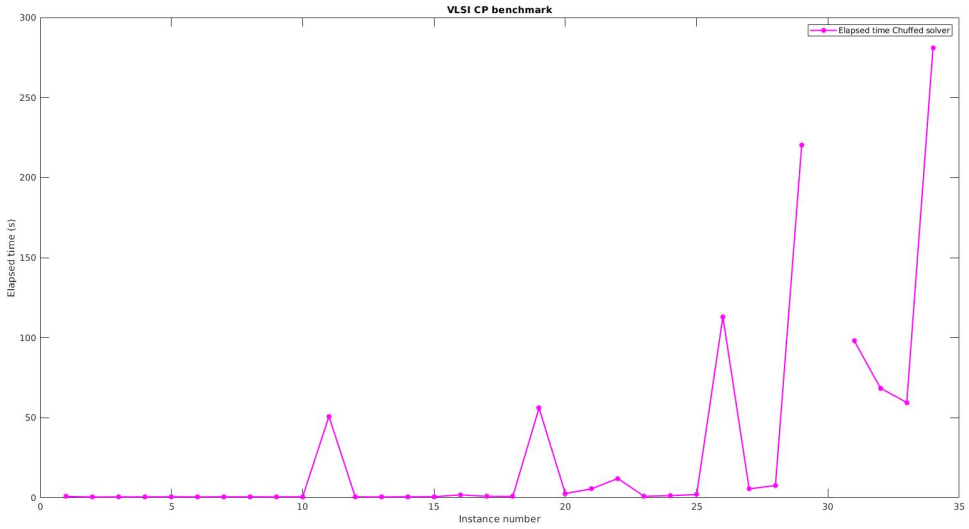
In this case, indomain_min, which assigns the variable its smallest domain value, was selected.

Once this basic search annotations were tried out, a sequential search constructor was utilized to specify the order of the search. In fact, the seq_search search constructor annotation first undertakes the search specified by the first annotation in its list, then, after that every variable in the annotation is fixed, it proceeds with the next search annotation.

A seq_search executing two int_search, the first on x, the second on y, was used leading to the best experimental results.

5 Results

As shown in the graph below, the model was capable to solve out of the 41 instances that were solved on a machine equipped with Intel Core i7-8550U CPU @ 1.80GHz.



Regarding the model allowing rotation, the following results were obtained:

6 Conclusions

Constraint Programming is the best way to solve this kind of problem as shown by the results. Clearly the model that allows rotation gives worse results as it is more complex to deal with.