Very Large Scale Integration design in CP

Combinatorial Decision Making and Optimization

Module 1

Alex Costanzino

alex.costanzino@studio.unibo.it

GitHub

Academic year 2020-2021

Abstract

Very Large Scale Integration (VLSI) refers to the trend of integrating more and more circuits into a single silicon chip. So far this trend has been depicted by the famous and empirical Moore's law, first devised in the 1965 [2].

The modern trend of shrinking transistor sizes, allowing engineers to fit more and more transistors into the same area of silicon, has pushed the integration of more and more functions of cellphone circuitry into a single silicon die. Nevertheless, Moore's law seems to have reached its limits, due to the dissipative phenomena of semiconductors, posing another challenge, not only in the optimization of the space, but also in the optimization of the local energy on the plate [3].

In any case, space optimization enabled the modern devices to mature into a powerful tool that shrank from the size of a large brick-sized unit to a device small enough to comfortably carry in a pocket or purse, with a video camera, touchscreen, and other advanced features.

Contents

Project work

The main task is the following: 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 device is minimized, ensuring an higher portability.

Setup

The project was developed on the following machine:

- Intel® Core™ i7-8565U 6 Core @ 1.80 GHz;
- NVIDIA® GeForce MX110™ 2GB;
- DDR4 8 GB RAM.

The main softwares used to tackle the problem are:

- *MiniZinc 2.5.5*, a free and open-source constraint modeling language;
- *Python 3.8.11*, and some of its basic libraries.

Data

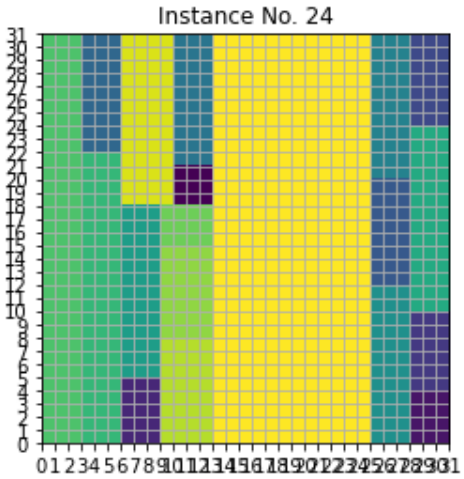
Within the task, also forty instances, with increasing complexity, have been made available to test the models. The instances are under the form of **.txt* files.

Utilities

Some utilities have been developed to make interact the underlying operative system with Minizinc and Python.

Instance converter Since the instances are provided as text, but Minizinc needs to be fed with its own data format, a converter has been developed. The utility simply reads the text file, line by line, converting them into a more suitable syntax for Minizinc, and then write them on a **.dzn* file.

Solution visualizer Since the outputs are also produced as text, a tool to obtain a graphical representation has also been created. The utility reads the text file, line by line, catch the data of the solution and use *Matplotlib* to visualize it.



Executor The executor script interacts with the command prompt in order to launch models via Minizinc, with some parameters, such as:

- Time limit;
- Type of solver;
- Models, inputs and solutions paths.

At the end of each instance, the script reads the output from Minizinc on the command prompt, clean it and produce a **.txt* file with a suitable format.

Plotter A routine that plots an histogram with elapsed times for each solved instances, in log scale for better readability.

1 Base model

For the base model, the parameters fixed by each instance, are:

- Number of integrated circuits to place on the plate: n (`ICs_number`);
- Width of the plate: W (`width`);
- Widths of the integrated circuits to place: w (`IC_widths`);
- Heights of the integrated circuits to place: h (`IC_heights`).

Other parameters, useful to limit the search space of the optimal height, are:

- An upper bound for the height of the plate: $B_{\text{up}} = \sum_{i=1}^{n} h_i$ (`upper_bound`);
- A lower bound for the height of the plate: $B_{\text{low}} = \min_{i \dots n} h_i$ (`lower_bound`).

The decision variables are:

- The height of the plate: $H \in [B_{\text{low}}, B_{\text{up}}] = \max_{i \dots n} y_i + h_i$ (`height`). Note that the domain is a kind of constraint itself: it is the maximum sum of position and height, over each components, namely the most closer to the border of the plate. In this manner the domain should be reduced to a discrete set of point (at most as the number of integrated circuits);
- The horizontal positions of the integrated circuits: x (`x`). For each horizontal position it holds $x_i \in [0, W - \min(w)]$. This is also a constraint itself, which states that the horizontal position of each integrated circuit cannot be outside of the plate;
- The vertical positions of the integrated circuits: y (`y`). For each vertical position it holds $y_i \in [0, B_{\text{up}} - \min(h)]$. This is also a constraint itself, which states that the vertical position of each integrated circuit cannot be outside of the upper bound of the plate.

The main constraints are:

- All integrated circuits shall fit on the silicon plate, taking into account not only the position but also the dimensions:

$$\forall i \in [1, n]$$
$$x_i + w_i \leq W$$
$$y_i + h_i \leq H$$

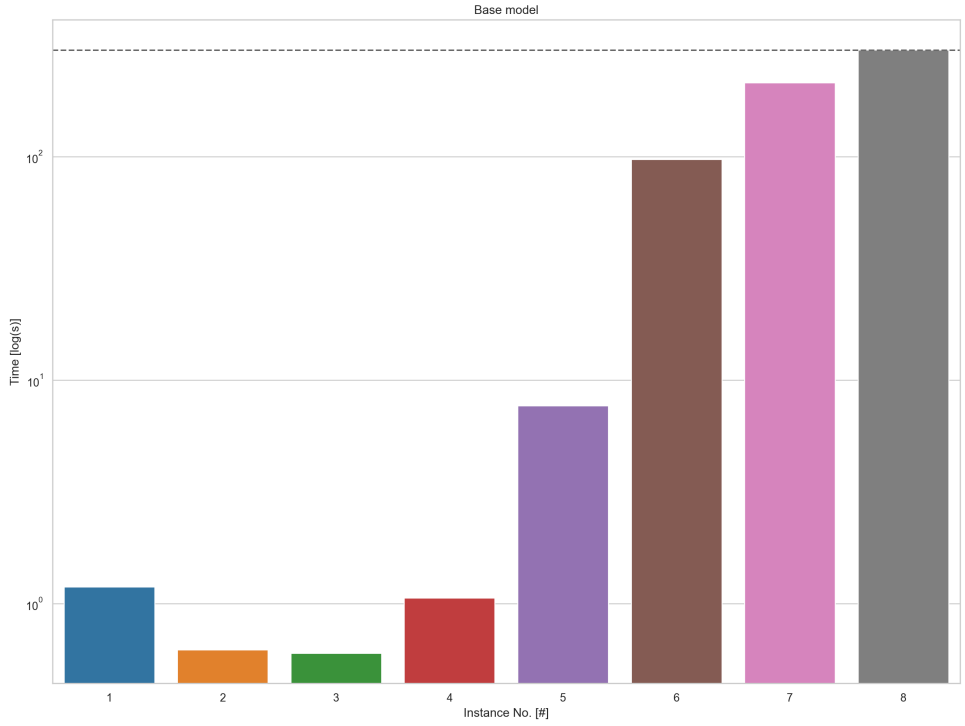
- All integrated circuits shall not overlap:

$$\forall i, j \in [1, n] : i \neq j$$
$$x_i + w_i \leq x_j \vee x_i - w_j \geq x_j \vee y_i + h_i \leq y_j \vee y_i - h_j \geq y_j$$

The goal of the problem is to place the circuits maintaining the lowest height possible for the silicon plate, therefore:

$$\min H$$

This basic model - with the *Gecode* - has been able to solve only the first seven instances within the time boundary of 300 seconds.

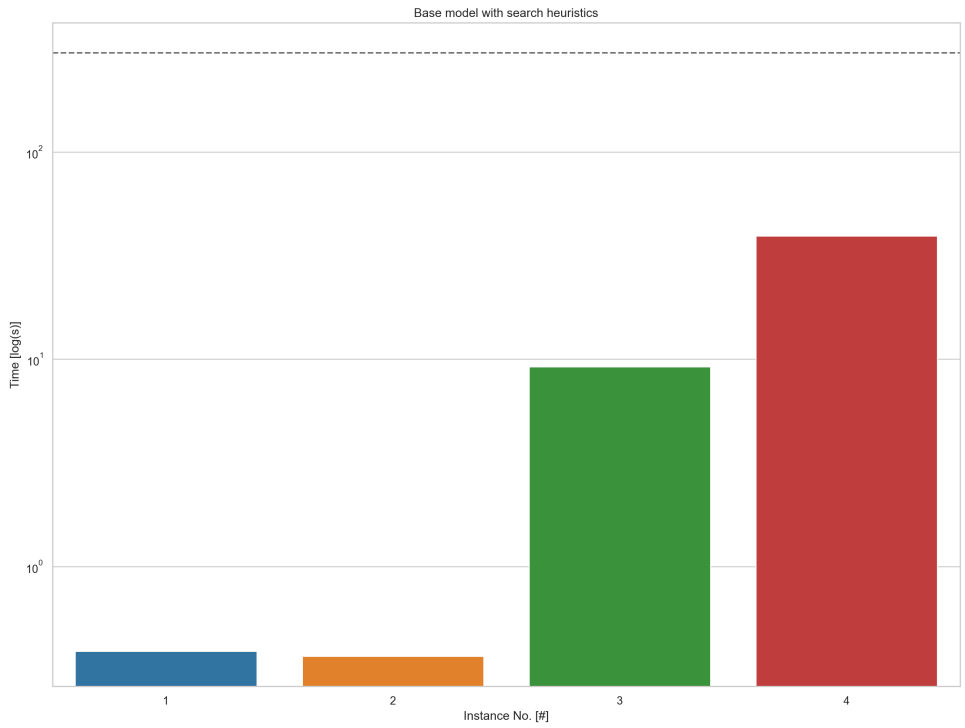


1.1 Base model with heuristics

With Minizinc it is possible to add search annotations in order to guide the solver. We can enrich the model with:

- Search heuristics: strategies to guide the solver in the search space;
- Restart heuristics: strategies to tune the intensification-diversification trade off.

With such simple model, the tested search annotations actually worsened the search, solving just four instances:



1.2 Base model with implied constraints

It is possible to enhance the model with the following consideration: in any solution, if we draw a horizontal line and sum the horizontal sides of the traversed circuits, the sum can be at most w. A similar property holds if we draw a vertical line.
These properties can be modeled as implied constraints. These constraints are logical consequences of the initial specification of the problem [5]. Though adding an implied constraint to the specification of a constraint satisfaction problem does not change the set of solutions, it can reduce the amount of search the solver has to do.
In our case, these implied constraints can be depicted as:

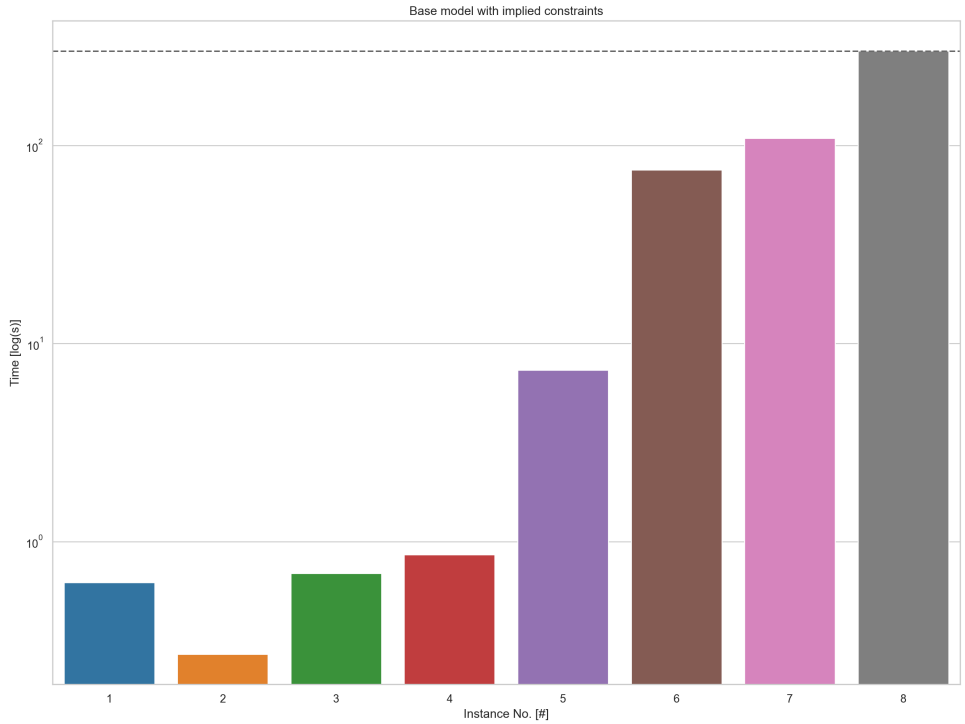
$$\forall i \in [1, n]$$

$$\max(x_i + w_i) \leq W$$

$$\max(y_i + h_i) \leq B_{\text{up}}$$

Intuitively, the integrated circuit chosen by the max function will be the the most closer to the edge of the plate, hence the sum between its coordinate and dimension will be the sum of all sides.

With such simple model, these implied constraints delivered a very slight improvement by temporal means:

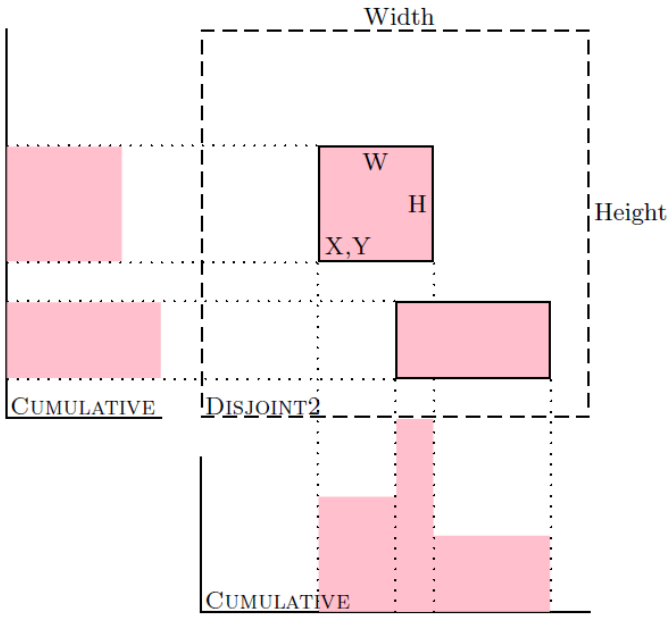


2 Global model

If we take into account the simultaneous presence of simple constraints, it is possible to further reduce the domains of the decision variables. These constraints encapsulating a set of other constraints are called global constraints [4]. In our case these constraints are expressed by a non-overlapping constraint in two dimensions and two (redundant, they behave as implied constraints) cumulative constraints that work on the projection of the packing problem in x and y direction [6].

In MiniZinc, we can substitute the non-overlap constraint with the `diffn` constraint, a generalised multidimensional non-overlapping constraint: it holds if, for each pair of orthotopes, the orthopes do not overlap [8]. An orthotope corresponds to the generalization of the rectangle and box to the n-dimensional case. In addition its sides are parallel to the axes of the placement space.

Two orthotopes do not overlap if one of the orthotopes has zero size or if there exists at least one dimension where their projections do not overlap.



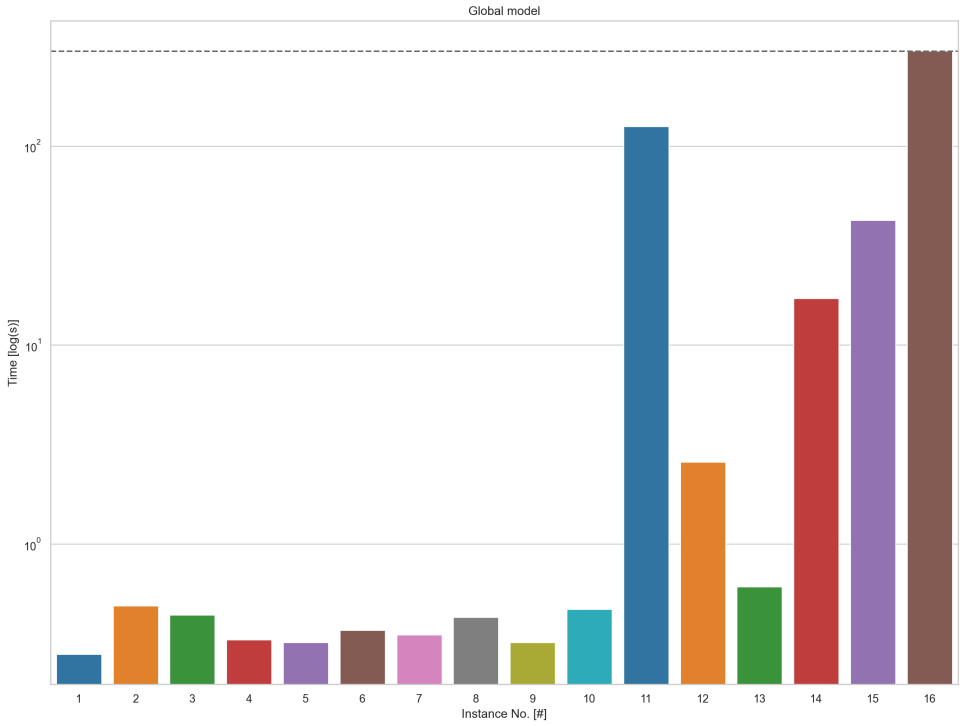
Moreover, we can use the two `cumulative` constraints, always available in Minizinc. Hence, the two new constraints are:

$$\text{diffn}(x, y, w, h)$$

$$\text{cumulative}(x, w, h, H)$$

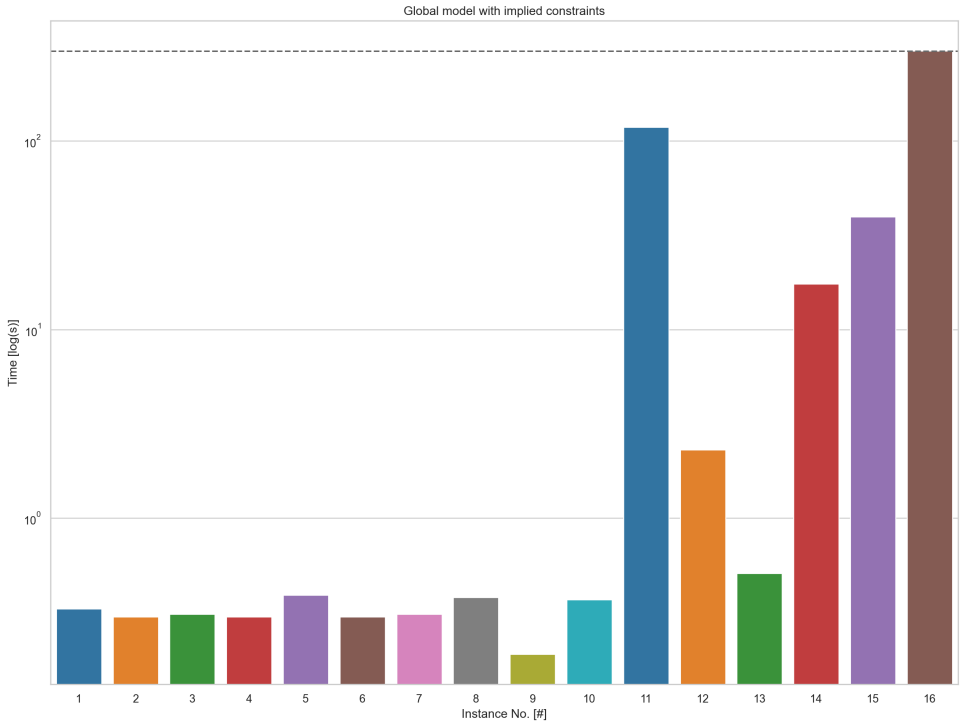
$$\text{cumulative}(y, h, w, W)$$

With this more elaborated model we are able to solve fifteen instances of the problem:



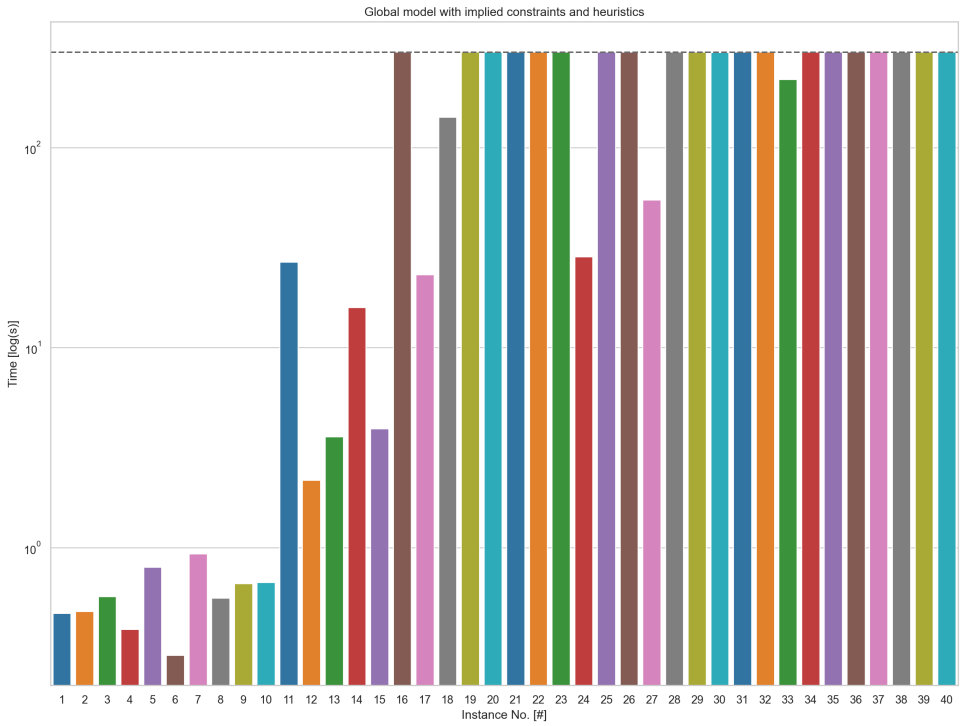
2.1 Global model with implied constraint

If implied constraints are kept, even better temporal performances can be achieved:



2.2 Global model with implied constraint and heuristics

If also heuristics are used, it is possible to solve up to twenty instances.



3 Symmetry-breaking constraints

A symmetry in a constraint satisfaction problem is a bijection that maps solutions to solutions and non-solutions to non-solutions. Symmetry in constraint programs can cause problems for an algorithm that searches a space of partial assignments due to redundancy in the search space [1]. One of the most popular methods for reducing symmetry is to add to the model extra constraints, so called symmetry-breaking constraints. In contrast to implied constraints, symmetry-breaking constraints are not logical consequences of the initial specification and adding them to a model may reduce the set of solutions [5].
To date, the choice of symmetry-breaking constraints has been determined solely by how much symmetry is broken, weighed against the cost of adding the symmetry-breaking constraints themselves [5]: a weaker symmetry-breaking scheme might sometimes be preferred to a stronger one because of the implied constraints that can be derived from it.
In this problem the main idea is to place the biggest component in the bottom left side of the silicon plate, and - in general - have an higher density of integrated circuits in that side of the board. Note that these constraints will likely have more effect when rotations will be allowed, since there will be more symmetries due to rotations.
From now on the model considered as baseline will be the one with global constraints and implied constraints.
To take into account symmetry-breaking constraints, other two parameters are added to the model:

- Index of the integrated circuit with the maximum height: $i^* = \arg\max_{i \dots n} h_i$ (`max_height_index`);
- Areas of the integrated circuits to place: $a_i = w_i h_i$ (`IC_areas`).

The symmetry-breaking constraints are modeled as follows:

- The position of the highest component shall be in the origin:

$$x_{i^*} = 0 \land y_{i^*} = 0$$

  Note that this is a rather strong constraint, it could actually worsen the performances.

- There shall be an higher density of components on the left side of the silicon plate:

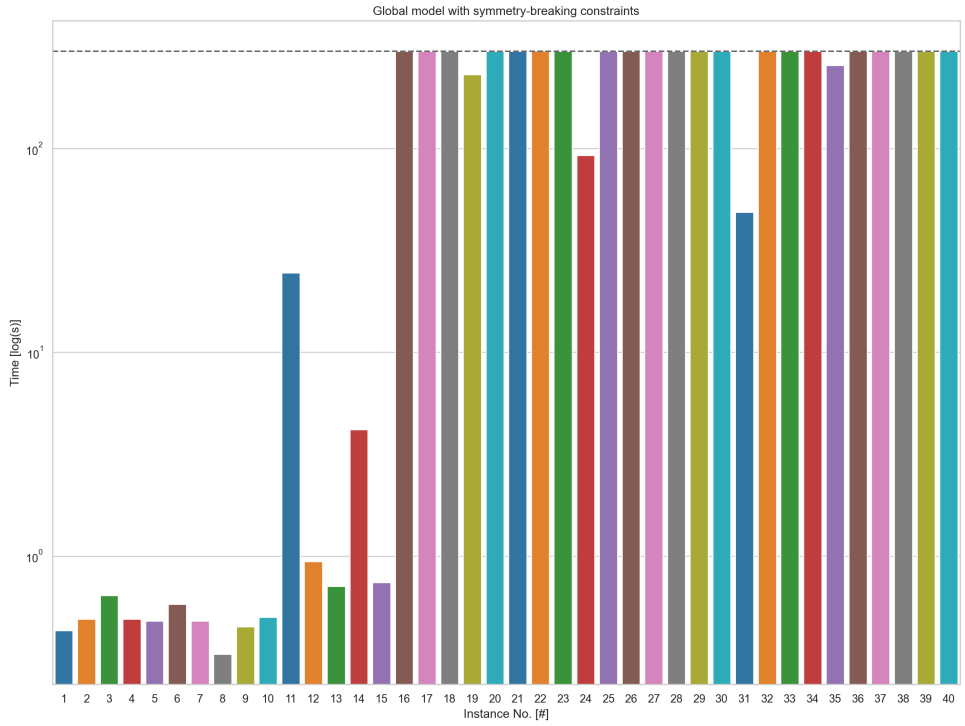
$$\sum_{i:x_i \leq \frac{W}{2}} a_i \geq \sum_{i:x_i > \frac{W}{2}} a_i$$

Since the first new constraint can be harder to optimized, it is possible to use a relaxed version of that constraint (it could also be totally discarded, but without rotations it would not improve the model so much), namely that the position of the highest component shall be in the first semi-plate [6]:

$$x_{i^*} \leq \frac{W}{2} \wedge y_{i^*} \leq \frac{H}{2}$$

3.1 Symmetry-breaking constraints with heuristics

Also in this case, if search and restart heuristics are considered, there is a good improvement in terms of temporal performances:



4 Rotations

To take into account the possible rotation of the integrated circuits, a new vector of boolean decision variables can be introduced. Each entry of the vector define the state of a piece, rotated or not rotated: r (`rotation`).
Consequently, the definition of the height of the plate slightly changes:

$$H \in [B_{\text{low}}, B_{\text{up}}] = \max_{i \dots n} \Big(y_i + r_i w_i + (1 - r_i) h_i \Big)$$

As far as concerns the definition of the constraints, the fitting ones change a bit:

$$\forall i \in [1, n]$$
$$x_i + r_i h_i + (1 - r_i) w_i \leq W$$
$$y_i + r_i h_i + (1 - r_i) w_i \leq H$$

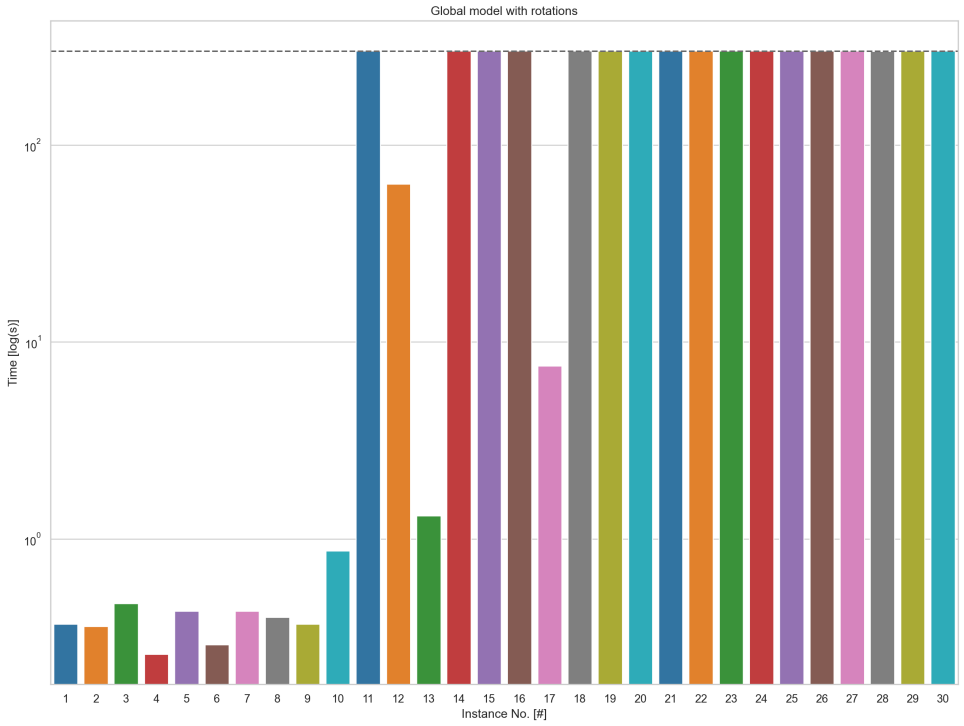
Also the implementation of the `diffn` changes slightly:

$$\text{diffn}(x, y, (1 - r_i) w_i, r_i h_i)$$

And a new implied constraint emerges, a circuit shall not rotate if its height is greater than the width of the plate, fixed by the instance:

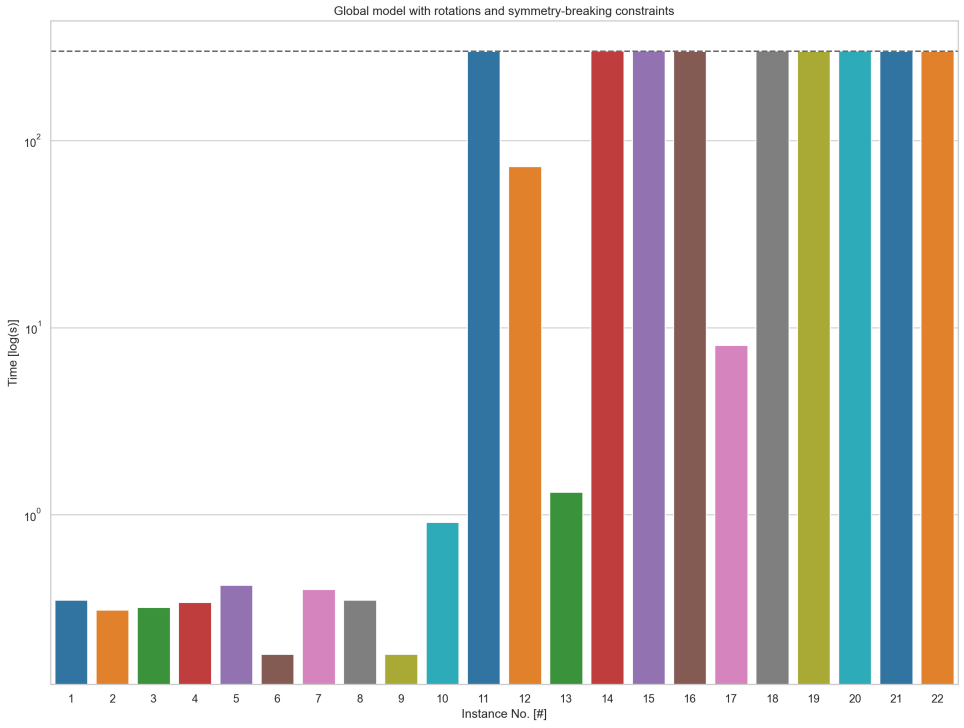
$$\forall i \in [1, n]$$
$$h_i > W \implies r_i = \text{F}$$

Of course, since taking into account also the rotations of the integrated circuits adds more complexity, the general performance achieved so far is worsened. Some instances are no longer solved and some others have drops in time performances:



4.1 Rotations with symmetry-breaking constraints

If symmetry-breaking constraints are enforced, there is a very slight improvement in some easy instances, while some others are a bit worsened:



5 Search and restart strategies

So far the search and restart heuristics have been chosen by chance in order to seek for an improvement in performances. Now, a more thoroughly test will be done.
For simplicity, only some instance will be taken as pivots, and tested with different search heuristics, in order to find a promising combination. Then, the most promising one will be selected to run over all instances. Of course running the test only on few instances does not guarantees to find the best combination, but since running tests is very time-consuming, this can be a good way to explore different combinations.
There are two kind of search heuristics that can imposed to the solver: variable choice (determines the order in which the variables are chosen for search) and the way to constrain.
The variable choice annotations used are:

- `input_order`: choose in order from the array;
- `first_fail`: choose the variable with the smallest domain size;
- `dom_w_deg`: choose the variable with the smallest value of domain size divided by weighted degree, which is the number of times it has been in a constraint that caused failure earlier in the search.

The ways to constrain a variable used are:

- `indomain_random`: assign the variable a random value from its domain;
- `indomain_split`: bisect the variables domain excluding the upper half.

Note that restart search does not make much sense if the underlying search strategy does not do something different the next time it starts at the top.
Any kind of depth first search for solving optimization problems suffers from the problem that wrong decisions made at the top of the search tree can take an exponential amount of search to undo. One common way to ameliorate this problem is to restart the search from the top, thus having a chance to make different decisions.
For the restart strategies, the parametrization chosen will be one suggested by MiniZinc documentation [9]:

- `restart_constant(100);`

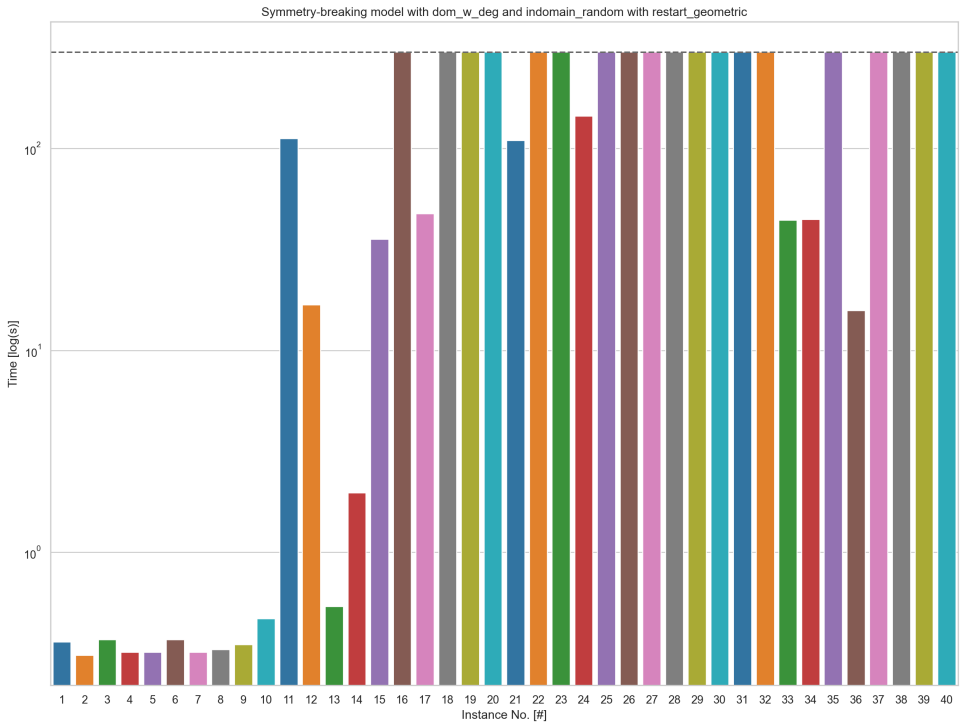
- `restart_linear(100)`;
- `restart_geometric(1.5,100)`;
- `restart_luby(100)`.

5.1 Global model with symmetry-breaking constraints

For the global model, the most promising combinations are:

- `input_order` and `indomain_split` with `no_restart`;
- `dom_w_deg` and `indomain_random` with `no_restart`.

Even if the first combination presented the best temporal performance over all combinations, the `dom_w_deg` heuristics showed consistent improvements despite the restart strategy adopted.

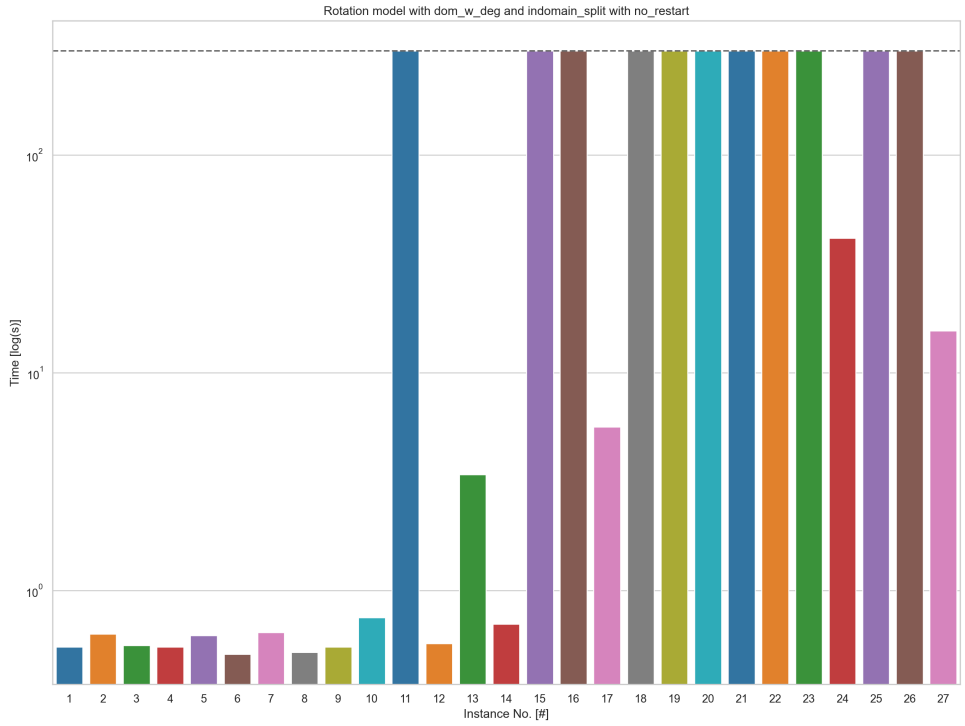


5.2 Rotation model

In general - for rotation model - restart does not seems to be a winning strategy. Indeed, the best combinations have been:

- `dom_w_deg` and `indomain_split` with `no_restart`;

- `first_fail` and `indomain_split` with `no_restart`.



6 Final remarks and possible future developments

The best model so far has been - as expected - is the one with global constraints, implied and symmetry-breaking constraints, with proper search and restart heuristics. On its own this model has been able to solve up to 21 instances.
In general, taking into account also other deployed models, in CP a total of 25 instances have been solved.
Further improvements may be achieved with:

- Use of decomposed constraints to speed-up the solver;
- Use of a dual model, where a matrix, describing the temporary placement of each integrated circuit, is instantiated. In this kind of representation more effective symmetry-breaking constraints may be enforced;
- Channeling with the aforementioned model;
- Implementation of Lodi's model [7].

References

[1] James Crawford et al. "Symmetry-Breaking Predicates for Search Problems". In: Morgan Kaufmann, 1996, pp. 148–159.

[2] R.R. Schaller. "Moore's law: past, present and future". In: *IEEE Spectrum* 34.6 (1997), pp. 52–59. DOI: 10.1109/6.591665.

[3] Kaustav Banerjee, Massoud Pedram, and Amir H. Ajami. "Analysis and Optimization of Thermal Issues in High-Performance VLSI". In: *Proceedings of the 2001 International Symposium on Physical Design*. ISPD '01. Sonoma, California, USA: Association for Computing Machinery, 2001, pp. 230–237. ISBN: 1581133472. DOI: 10.1145/369691.369779. URL: https://doi.org/10.1145/369691.369779.

[4] Jean-Charles Régin. "Global Constraints and Filtering Algorithms". In: (Jan. 2003). DOI: 10.1007/978-1-4419-8917-8_4.

[5] Alan M. Frisch, Christopher Jefferson, and Ian Miguel. "Symmetry Breaking as a Prelude to Implied Constraints: A Constraint Modelling Pattern". In: *Proceedings of the 16th European Conference on Artificial Intelligence*. ECAI'04. Valencia, Spain: IOS Press, 2004, pp. 171–175. ISBN: 9781586034528.

[6] Helmut Simonis and Barry O'Sullivan. "Using Global Constraints for Rectangle Packing". In: (Jan. 2008).

[7] Vanessa Bezerra et al. "Models for the two-dimensional level strip packing problem – a review and a computational evaluation". In: *Journal of the Operational Research Society* 71 (Apr. 2019), pp. 1–19. DOI: 10.1080/01605682.2019.1578914.

[8] Sophie Demassey. *Global Constraint Catalog*. 2020. URL: https://sofdem.github.io/gccat/gccat/Cdiffn.html.

[9] Peter J. Stuckey, Kim Marriott, and Guido Tack. *2.5.4. Restart*. 2020. URL: https://www.minizinc.org/doc-2.5.5/en/mzn_search.html#restart.