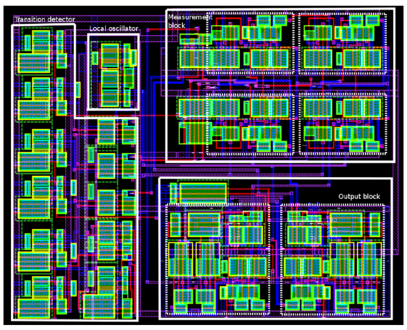
ALMA MATER STUDIORUM · UNIVERSITY OF BOLOGNA

VLSI (Very Large Scale Integration) design optimization problem: CP solution



Combinatorial Decision Making and Optimization

Giuseppe Murro (giuseppe.murro@studio.unibo.it)
Giuseppe Boezio (giuseppe.boezio@studio.unibo.it)
Salvatore Pisciotta (salvatore.pisciotta2@studio.unibo.it)

August 28, 2021

Contents

1 Introduction

VLSI (Very Large Scale Integration) refers to the trend of integrating circuits into silicon chips. A typical example is the smartphone. 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 (i.e. plate). This enabled the modern cellphone 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. The aim of this project is to design a solution for the problem of put all the given input circuits in the plate optimizing its height.

2 Instances

2.1 Input

Input instances are presented using the following variables:

- w = width of the silicon plate
- n = number of circuits to insert in the plate
- x_i = horizontal dimension of i^{th} circuit
- y_i = vertical dimension of i^{th} circuit

where an instance is written in the following way:

w
n
x_0 y_0
x_1 y_1
...

2.2 Output

Starting from input instances data, once that the optimization process is finished it is given the output as:

w l
n
x_0 y_0 px_0 py_0
x_1 y_1 px_1 py_1
...

where:

- w, n, y_i, x_i are the same of the input
- l = maximum height reached by the circuits configuration
- px_i = horizontal left-down point coordinate of the i^{th} circuit block
- py_i = vertical left-down point coordinate of the i^{th} circuit block

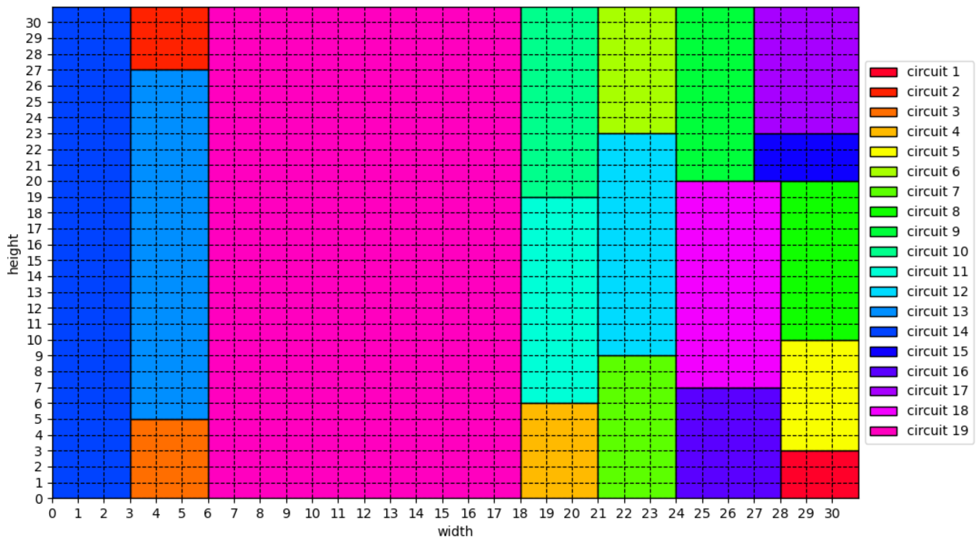


Figure 2.1: Output of an instance with w = 31 and n = 19

3 Modelling

3.1 Preliminary Reasoning

3.1.1 Modelling variables

The first part of the modelling process is the one of deciding how to model the problem and define the variables to use. Since the output to produce requires the left-bottom point of each block, it was decide to use two arrays that describe respectively the x and y coordinate of the blocks maintaining the order of the circuits as the input in order to easily find the correlated length and width. Both the arrays have n length, with n representing the number of circuits in input. The domain of the px vector (vector of the x coordinates) goes from 0 to w - min(x), while the py vector (vector of the y coordinates) has a domain from 0 to $l_{max} - min(y)$, where min(y) is the minimum value of the lengths of the input circuits while, l_{max} computed as:

$$l_{max} = \begin{cases} max\_y & \frac{(\sum_{i=1}^{n} y_i) \cdot max\_x}{w} < max\_y \\ \frac{(\sum_{i=1}^{n} y_i) \cdot max\_x}{w} & \text{otherwise} \end{cases} \tag{3.1}$$

Where max_x and max_y are respectively the maximum value of lengths and heights of the input circuits. The choice of these possible values has been done taking in consideration the unluckiest case in which all circuits dimensions are max_x and max_y, so l_{max} is computed calculating the number of circuits that can be fitted horizontally and then how many rows of these blocks are needed vertically. But sometimes this approximation is not feasible if the result is less then max_y, so in that case could be used max_y as value for l_{max}.

3.1.2 Variable to minimize

As indicated in the introduction chapter, the aim of the project is to build a model that can minimize the height of the plate putting on it all the requested circuits blocks. So it is necessary to use a variable l that describes the height of the plate. Based on the variables previously described we can say that l is the maximum value of all the values in the vector py respectively summed with the height of the correspondent circuit.

3.1.3 Symmetries

Looking at the feasible configuration of optimal solutions it was discovered that configurations of the filled plate with the same l value are equivalent, so there could be exponentially many equal solutions and some of them can be seen as symmetries. The possible symmetries that it has been detected are:

- Vertical flip
- Horizontal flip
- 180° rotation

3.2 Constraints

3.2.1 Main constraints

Using the variables that have been defined it was possible to impose the main constraints of the problem:

- **Parallelism with scheduling**: concerning the optimization of the space in the plate, it was thought to use the global constraint *cumulative*. Looking at the task as a resource usage problem each circuit has been considered as an activity whose duration is the vertical length, its amount of resources is equals to its horizontal length while the total number of resources is w width. The same reason can be done in the opposite sense, so each circuit has an activity whose duration is represented by the horizontal length and amount of resources is equals to its vertical length. The constraints are the followings:

$$cumulative(py, y, x, w)$$
$$cumulative(px, x, y, l_{max})$$

- **Overlapping**: To avoid the possible overlapping of circuits the main idea was put a relationship between the end and the beginning of each pair of circuits for both coordinates. But in a second moment, looking at the packing constraints explained in the Minizinc library, it was decided to use the global constraint $diffn$ that given the origin points and sizes of rectangles impose the non-overlapping among them:

$$diffn(px, py, x, y)$$

3.2.2 Implied constraints

There is one important implied constraint that must be taken into account, if we draw a horizontal line and sum the horizontal sides of the traversed circuits, the sum can be at most w at each length. It can be said the same thing for the vertical axes imposing the maximum as the l_{max} value. But, if we do not consider the rotation of the blocks it is possible to avoid this last constraint due to the construction of the l_{max} value. The implemented formulae are the followings:

$$max(px_1 + x_1,\ px_2 + x_2,\ ..,\ px_n + x_n) <= w$$
$$max(py_1 + y_1,\ py_2 + y_2,\ ..,\ py_n + y_n) <= l_{max}$$

3.2.3 Symmetry breaking constraints

In order to break the symmetries, also the ones described in 3.1.3, it was thought to put the circuit with the biggest height always on the coordinate $(0, 0)$, in this way the biggest circuit is forced to be always at the bottom left corner of the plate, rejecting all other configurations where this circuit is in another position (including 180° rotation and vertical flip of the solution).

$$px(max\_y\_index) = 0 \wedge py(max\_y\_index) = 0$$

where max_y_index is the index of the circuit with the maximum height.

To break other equivalent solutions, including the horizontal flip, it was implemented a constraint that forces the circuits whose coordinate px_i lies in the left half part of the plate to have the sum of the covered area greater or equal than the ones in the right half side. The implemented constraint is the following:

$$\forall i \in \{1..n\} : area_i = x_i * y_i$$
$$\sum_{i=1,\, px_i <= w\, div\, 2}^{n} area_i \;>=\; \sum_{i=1,\, px_i > w\, div\, 2}^{n} area_i$$

3.3 Dual model

3.3.1 Introduction

In addition to the aforementioned model, another one has been realized and used for implementing channelling and symmetry breaking in an equivalent way.
The idea was to label each coordinate with a number between 1 and $n + 1$ (n = amount of circuits). The number is used to identify which circuit is on that coordinate, $n + 1$ whether there is not a circuit on it. We consider coordinates which go from $(0, 0)$ to $(l_{max} - 1, w - 1)$ because the effective dimension of the plate is not fixed and depends on the instance.

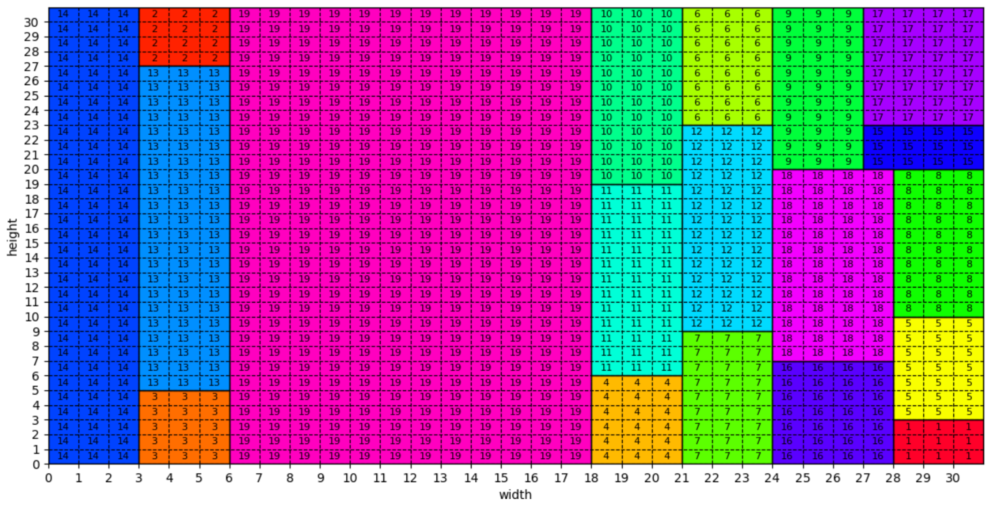


Figure 3.1: Output of an instance with w = 31 and n = 19

3.3.2 Channelling

The matrix of coordinates is called **plate** and is combined with the previous model in the following way:

$$\forall i \in \{0..l_{max} - 1\}, j \in \{0..w-1\}, k \in \{1..n\}:$$
$$board_{ij} = k \iff px_k <= i \wedge i < px_k + x_k \wedge py_k <= j \wedge j < py_k + y_k$$

3.3.3 Constraints

The constraint **global_cardinality_low_up** allows to impose for each value that the whole amount of occurrences in the first array is bounded between low and upper bound provided as input. In this case the constraint is used to impose a global cardinality value for each circuit in rows and columns. Constraints have been implemented as follows:

$$\forall i \in \{0 \,..\, l_{max} - 1\}:$$
$$global\_cardinality\_low\_up([board_{i0}, board_{i1}, ..., board_{i,w-1}], 0..n, [0_0, 0_1, ..., 0_n], [w_0, w_1, ..., w_n])$$
$$\forall j \in \{0 \,..\, w - 1\}:$$
$$global\_cardinality\_low\_up([board_{0j}, board_{1j}, ..., board_{w-1,j}], 0..n, [0_0, 0_1, ..., 0_n], [l_{max_0}, l_{max_1}, ..., l_{max_n}])$$

3.3.4 Symmetry breaking

Symmetry breaking constraints take into account the following symmetries with related formulae implemented using **lex_lesseq** ordering constraints:

- **Horizontal Flip**

  $\forall i \in \{0 \,..\, l_{max} - 1\}, j \in \{w - 1 \,..\, 0\}: lex\_lesseq([board_{00}, board_{01}, ..., board_{l_{max-1}w-1}], [board_{ij}])$

- **Vertical Flip**

  $\forall i \in \{l_{max} - 1 \,..\, 0\}, j \in \{0 \,..\, w - 1\}: lex\_lesseq([board_{00}, board_{01}, ..., board_{l_{max-1}w-1}], [board_{ij}])$

- **180° Rotation**

  $\forall i \in \{l_{max} - 1 \,..\, 0\}, j \in \{w - 1 \,..\, 0\}: lex\_lesseq([board_{00}, board_{01}, ..., board_{l_{max-1}w-1}], [board_{ij}])$

3.4 Rotation model

3.4.1 Introduction

A variation of the problem consists of allowing a rotation of circuits. This means that the final height and length of each circuit is not necessary equals to dimensions provided in input but they could be swapped.

3.4.2 Constraints

Starting from this assumption, a new array of boolean **rotation** has been added to the previous model to express the fact that original dimensions of the i^{th} circuit have been swapped or not. From a logical point of view, constraints are the same of the previous model since dimension of each circuits are not taken into account as provided in input but how they are actually used. For this reason we have introduced two new arrays storing real height and width of each circuit and they are related to the rotation array in the following way:

$$\forall i \in \{1 \mathinner{..} n\}:$$

$$x\_r_i = \begin{cases} y_i & \text{if } rotation_i \\ x_i & \text{otherwise} \end{cases} \tag{3.2}$$

$$y\_r_i = \begin{cases} x_i & \text{if } rotation_i \\ y_i & \text{otherwise} \end{cases} \tag{3.3}$$

where x_r and y_r are the real circuit dimensions whereas x and y are dimensions provided in input.

It can be noticed that when a circuit has an height greater than w, it cannot be rotated. Therefore the i^{th} circuit will have $rotation_i$ equal to false.

$$\forall i \in \{1..n\}$$
$$y_i > w \implies rotation_i = False$$

3.4.3 Output

Output is the same of the initial model but it was added to each row a letter R whether the circuit has been rotated. Following an example:

w l
n
$x_0\ y_0\ px_0\ py_0$
$x_1\ y_1\ px_1\ py_1\ R$
...

4 Search heuristic and restart

4.1 Search heuristic

To compare the performances of the model, we have used different search heuristic both for variables and domains.

Variable Search Heuristics:

- input_order
- first_fail
- dom_w_deg

Domain Search Heuristics

- indomain_min
- indomain_random

Some combinations variable/domain heuristics have been considered for a comparison.

4.2 Restart

Because of heavy tail behaviour, it was needed to introduce randomness to break a deterministic behaviour in solution searching operation. For this reason restart has been implemented in different ways:

- restart_constant(100)
- restart_linear(100)
- restart_geometric(1.5,100)
- restart_luby(100)

All arguments have been taken as empirically efficient from Minizinc official guide.

5 Results

5.1 Final model

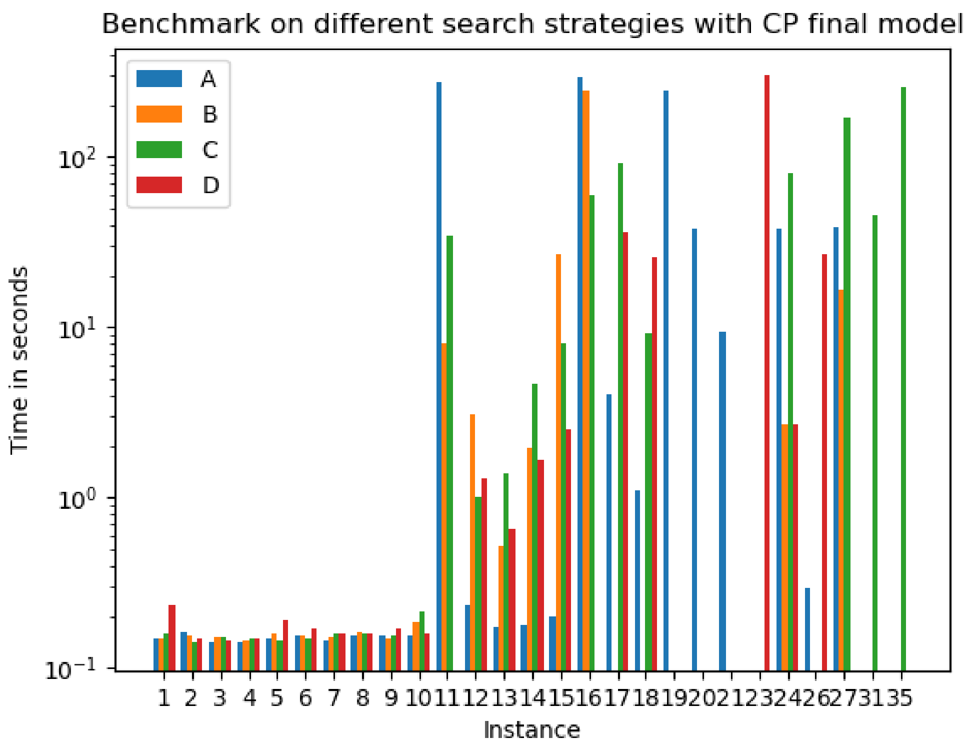


Figure 5.1: **A**: domWdeg, min domain without restart **B**: domWdeg, random domain and linear restart, **C**: domWdeg, random domain and luby restart, **D**: dom, random domain and linear restart

As it can be seen in 5.1 the first ten instances can be solved independently from the search while starting from the 11th instance the combination of the search and restart method might influence the production of a feasible result. Model A, C and D can produce results with an high number of circuits and width, for this reason they are the most performing even if they do not solve always the same instances. For display purposes only results of the best models have been shown, other search strategies have been executed but they had produced worse results than models in 5.1

In the table 5.1 below, we can see the results of some different search method applied at the same instance, the instance 24, using the final model. The outcomes shown are the ones that produce acceptable results so other combination of heuristics and restart was made but not reported in the table. The execution was made using MiniZinc solver Gecode 6.3.0, imposing as time limit 300 seconds.

Heuristic var	**Heuristic dom**	**Restart**	**n^{o}solutions**	l **value**	**Failures**	**Restarts**	**Solving time**
domWdeg	**min**	**no restart**	4	31	2178524	0	53s
domWdeg	**min**	**luby**	2	33	13027228	19451	300s
domWdeg	**min**	**linear**	35	32	12225976	63	300s
domWdeg	**min**	**geometric**	4	31	5594021	28	134s
domWdeg	**random**	**luby**	32	31	4648301	8221	118s
domWdeg	**random**	**no restart**	19	73	16596362	0	300s
domWdeg	**random**	**linear**	37	31	134201	87	4s
first fail	**min**	**linear**	2	33	11415039	480	300s
first fail	**min**	**luby**	2	33	12073665	17645	300s
first fail	**random**	**linear**	25	31	141184	77	3s
first fail	**random**	**luby**	2	33	12088930	17664	300s
input_order	**random**	**linear**	45	45	19150658	664	300s
input_order	**random**	**luby**	49	37	18450954	27694	300s

Table 5.1: Result of the execution of final model on instance 24 using different search method

5.2 Rotation model

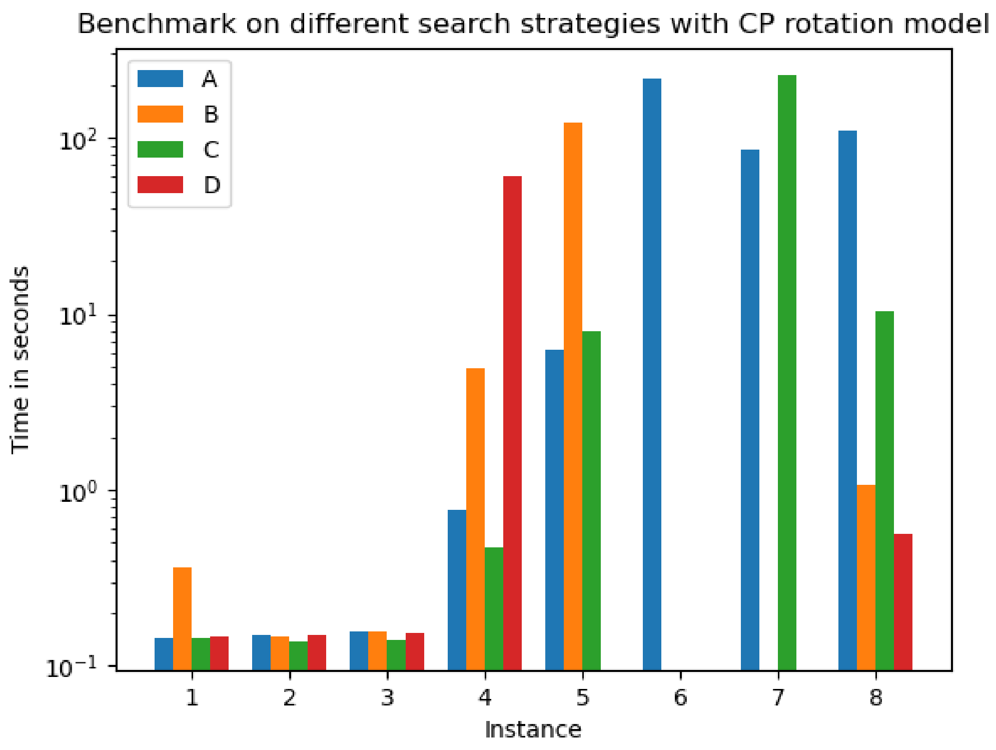


Figure 5.2: **A**: domWdeg, min domain without restart **B**: domWdeg, random domain and luby restart, **C**: domWdeg, random domain without restart, **D**: dom, random domain and linear restart

As expected solve instances with rotation model is more complex than with the final model, in terms of number of solved instances. Indeed, the number of optimal solutions for different instances is less than in the previous model despite the different search methods used. It can be noticed that the search mode A is the best one because it solves the highest amount of instances although the solving time is elevated.

In the table 5.2 below, we can see the results of some different search method applied at the same instance, the instance 8, using the rotation model. Also in this case the outcomes shown are the ones that produce acceptable results so other combination of heuristics and restart was made but not reported in the table.The execution was made using MiniZinc solver Gecode 6.3.0, imposing as time limit 300 seconds.

Heuristic var	**Heuristic dom**	**Restart**	**n^{o}solutions**	l **value**	**Failures**	**Restarts**	**Solving time**
domWdeg	**min**	**no restart**	12	15	8391031	0	146s
domWdeg	**min**	**luby**	4	24	13525895	18473	300s
domWdeg	**min**	**linear**	2	15	3	3	0.001s
domWdeg	**random**	**luby**	19	15	24769	592	4s
domWdeg	**random**	**no restart**	18	15	824084	0	13s
domWdeg	**random**	**linear**	11	15	536	14	0.012s
first fail	**min**	**no restart**	12	15	14415465	0	246s
first fail	**min**	**luby**	4	24	15239749	21375	300s
first fail	**min**	**linear**	1	30	17783168	597	300s
first fail	**random**	**no restart**	15	15	371252	0	6s
first fail	**random**	**linear**	20	15	86587	274	1.5s
first fail	**random**	**luby**	20	15	414189	110	6.5s
input_order	**random**	**no restart**	10	15	2962	0	0.05s
input_order	**random**	**luby**	13	15	2109	25	0.04s
input_order	**random**	**linear**	13	15	14421	29	0.3s
input_order	**min**	**linear**	4	15	0	3	0.001s
input_order	**min**	**no restart**	4	15	7	7	0.001s
input_order	**min**	**luby**	4	15	0	3	0.001s

Table 5.2: Result of the execution of rotation model on instance 8 using different search method

5.3 Comparison among different models

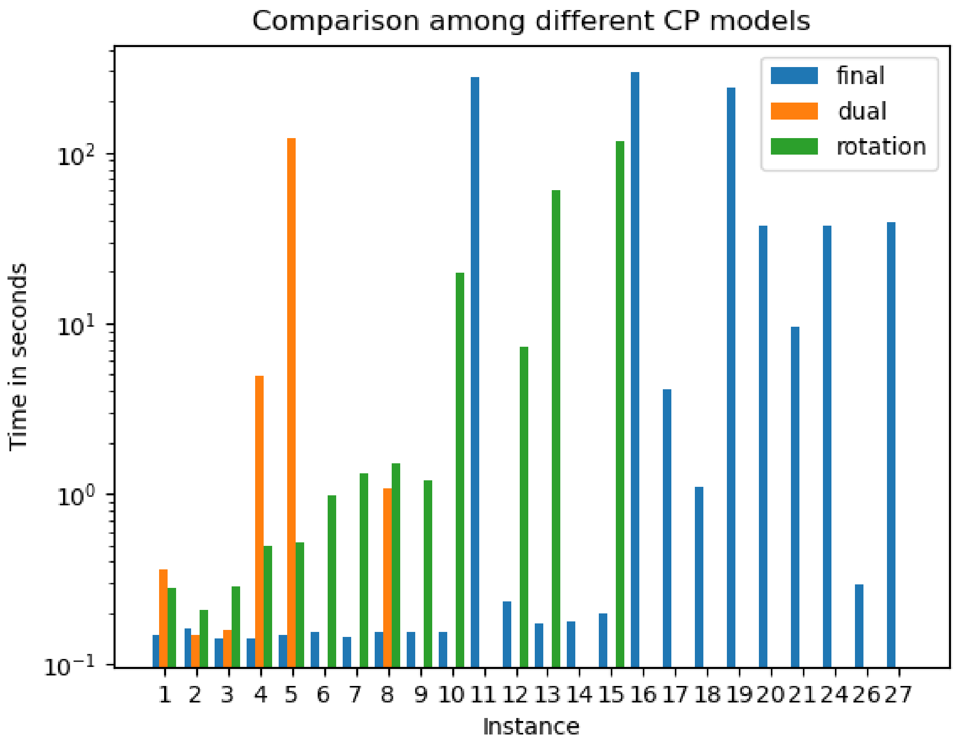


Figure 5.3: Comparison among different models

For the comparison it was used domWdeg and minimum domain with no restart as search method because, as previously shown, it produced the best results for final and rotation model. The search strategy has been used also for dual model. It can be noticed that for some instances it produces better results with respect to the rotation model.