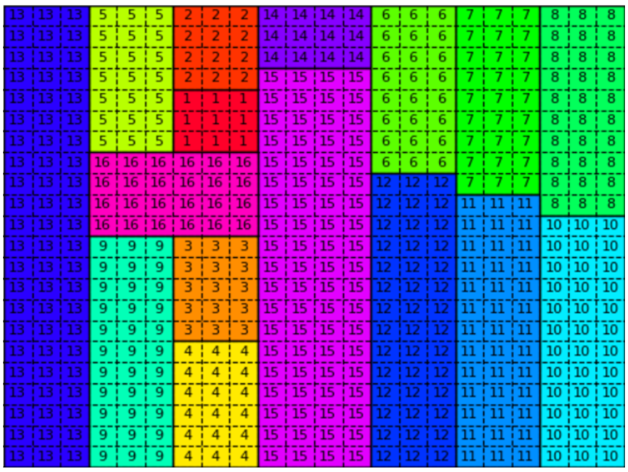
ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

Combinatorial Decision Making Optimization - Mod1

VLSI Project

Marco Costante - marco.costante@studio.unibo.it,
Alessandra Stramiglio - alessandr.stramiglio@studio.unibo.it

November, 2021



Academic Year 2020/2021

Contents

1 Introduction

The following project work applies for the first module of the Combinatorial Decision Making and Optimization course of the academic year 2020/2021. It demands to model and solving a combinatorial decision problem with Constraint Programming (CP) and SAT or SMT. The request is to define the VLSI of the circuits defining the electrical device. Given a fixed-width plate and a list of rectangular circuits, decide how to place them on the place so that the length of the final device is minimized.

2 Approach

In order to approach to the problem we reasoned on it so as to achieve a model which met our expectations. We started by defining a base model and then improving it. Then, we adapted the model with the purpose of considering also the rotation of the circuits. Before starting with the presentation of the model it's necessary to give a little description of the python scripts which allowed us to work and evaluate our model.

- *converter.py* to read the given instances in *.txt* files and write them as *.dzn*;
- *plots_solution.py* to visualize the solution, so how the circuits are arranged on the plate;
- *solve.py* to run the model on all the instances by calling *minizinc* command

Once we obtained a satisfying CP model, we translated it to SMT.

3 CP

3.1 Base Model

3.1.1 Data representation

Each instance of VLSI is encoded in a *.dzn* file as follows:

```
width = 8;
n = 4;
DX = [3, 3, 5, 5];
DY = [3, 5, 3, 5];
```

where:

- *width* represents the width of the silicon plate;

- n is the number of necessary circuits to place inside the plate;
- DX and DY are two arrays, indexed from *1* to n, containing the horizontal and vertical sizes of each circuit (for example the first circuit is of size 3x3).

Our goal is to minimize the height of the plate which becomes the objective function. It's calculated as follows:

$$height = max([Y[i] + DY[i] \mid i\ in\ 1..n]);$$

The solution will be encoded using two arrays of decision variables X and Y, indexed from 1 to n. Each element i of the X (resp. Y) array will contain the x (resp. y) bottom-left corner coordinate of the i-th circuit. The array X has values which go from 0 to the width of the plate minus 1, (resp. the array Y has values which go from 0 to the maximum heigh minus 1). Then it also indicate the length of the plate l.

```
array[N_CIRCUITS] of var 0..width-1: x;
array[N_CIRCUITS] of var 0..sum(DY)-1: y;
```

An example of solution may be:

```
14 14
9
3 3 11 7
3 4 11 10
3 5 8 0
3 6 5 0
3 7 11 0
3 8 5 6
3 9 8 5
5 4 0 0
5 10 0 4
```

A visualization of such solution is shown in figure 1.

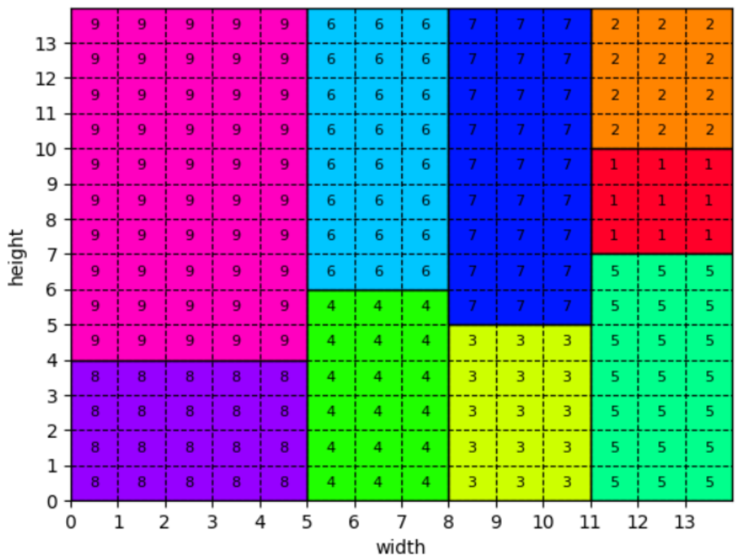


Figure 1: Graphical representation of a possible solution

3.1.2 Domain Reduction

In order to reduce the search space of the solver, it's a good practice to reduce the domain of the variables. We can define a possible range for both the coordinates of the bottom left corner of each rectangle. Each variable of the array x cannot assume a value greater than the width of the plate minus its width. Otherwise the circuit would fall out of the plate. Same reasoning can be applied to the array y. Thus, this domains constraint can be translated as follows:

```
constraint forall(i in N_CIRCUITS) (x[i]<=width-DX[i])::domain;
constraint forall(i in N_CIRCUITS) (y[i]<=height-DY[i])::domain;
```

3.1.3 Global constraints

Global constraints represent high-level modeling abstraction, they guarantee efficient inference algorithms for solvers. With the intention to describe the cumulative resource usage, in this case to respect container boundaries, we used the global constraint used for representing the resource usage in task scheduling.

In the cumulative constraint is required that a set of tasks given by start time s, durations d, and resource requirements r, never require more than a global resource bound b at any one time.

In our case we have to split the problem for both the x axis and the y axis. The task is the circuit, the start times is the x coordinate, the duration is DX and the global resource bound b is given by the *width* of the plate. In the other case the task is still the circuit but the other parameters will be: y coordinate, DY and the resource bound is the *height*.

```
constraint cumulative(y, DY, DX, width);
constraint cumulative(x, DX, DY, height);
```

Furthermore, to avoid the possible overlapping of circuits the main idea was to 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. This suits perfectly our problem of positioning circuits.

```
constraint diffn(X, Y, DX, DY);
```

3.2 Search heuristic

In Minizinc, search annotation allows to specify the policy adopted for finding a solution. By default no search strategy is defined and this leaves the search completely up to the underlying solver. It's important to point out that the search strategy is not really part of the model. Indeed, it's not required that each solver implements all possible strategies. The computational cost depends on the search tree. The shape of the tree is ruled by the propagation behavior, which is a property of the constraint solver.

It's possible to decide to run the solver with no annotation about the search:

```
solve minimize height;
```

Or by specifying it by choosing also the type of heuristic.

```
int: SEARCH_TYPE = 1;
solve :: search_ann
         minimize height;
```

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;

- dow_w_deg.

Domain search heuristics:

- indomain_min;
- indomain_random.

We have considered some combination to have a comparison.

3.3 Restart

Any kind of depth first search for solving optimization problems suffers from the problem that wrong decision made at the top of the search tree can take an exponential amount of search to undo. Since that, it is important to introduce randomness. Restart search is robust in finding solution because it can avoid getting stuck in a non-productive area of the search. The different restart annotations control how frequently a restart occurs. For this reason restart has been implemented in different ways:

- restart_constant(750)
- restart_linear(20)
- restart_geometric(1.5,500)
- restart_luby(250)

In order to apply restart it's necessary to specify the indicated annotation.

```
int: RESTART_TYPE = 1;
solve :: restart_ann
         minimize height;
```

3.4 Symmetry breaking

The solver may explore many symmetric variant of the same solution. Thus, to reduce the number of solutions it's good practice to apply symmetry breaking constraints. Symmetry breaking constraint in its simplest form, involves adding constraints to the model that rule out all symmetric variants of a (partial) assignment to the variables except one. Those constraints are called static symmetry breaking constraints. The basic idea behind symmetry breaking is to impose an order.

In our case we decided to place always the biggest circuit in the bottom left part of the plate. Moreover the second one biggest has to be placed always on the right and/or on the top of the biggest one. First of all we need to define an order of the circuits. This is done by sorting them in descending order considering their area, given by $DY \cdot DX$.

```
array[N_CIRCUITS] of int : ordered_circuits =
sort_by(N_CIRCUITS, [-DY[c]*DX[c] | c in N_CIRCUITS]);
```

After that, we can put the constraint considering the dimension of the circuits. To do that we use the global constraint *lex_lesseq* on the coordinates of the circuits. This constraint requires that the array x is lexicographical less than or equal to array y. It compares them from first to last element, regardless of indices.

It's possible to use the lexicographically order since we have already used *diffn* to make sure that each instance has different coordinates. However, it may happen that one of the two coordinates is shared between two circuits. This is why it's needed the less equal operator and not the the less not equal. In the latter case this would have been and hard constraint.

$$\forall i \in \{1..n\} : area_i = x_i * y_i$$

$$(y_{c1}, x_{c1}) \leq (y_{c2}, x_{c2}) \wedge x_{c1} * 2 \leq width \wedge y_{c1} * 2 \leq height$$

```
constraint symmetry_breaking_constraint(
  let {
    int: c1 = ordered_circuits[1], int: c2 = ordered_circuits[2]
  } in lex\_lesseq([y[c1],x[c1]], [y[c2],x[c2]]) /\
  x[c1] * 2 <= width /\ y[c1] * 2 <= height);
```

3.5 Rotation

In the first formulation of the problem the rotation of the circuits is not allowed. If we want to take it into account, it's necessary to introduce some modifications. In the data representation we add an array of boolean in which each index refers to a circuit and it's specified if it's rotated or not.

```
array[N_CIRCUITS] of var bool: is_rotated;
```

Thenceforth it's necessary to compute the actual width and height for each circuit. Indeed, if the circuit is rotated, DX and DY will be switched.

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

$$DX\_R = \begin{cases} DY_i & \text{if } is\_rotated_i \\ DX_i & \text{otherwise} \end{cases}$$

$$DY\_R = \begin{cases} DX_i & \text{if } is\_rotated_i \\ DY_i & \text{otherwise} \end{cases}$$

Taking into account rotation brings in new constraints to take care of. Some circuits may be excessively high, so they cannot be rotated.

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

3.6 Result and perfomances

3.6.1 Heuristic comparison

Final model - instance no. 33							
Variable heuristic	Domain heuristic	Restart heuristic	Propagations	Failures	Restarts	Solutions	Time (s)
input_order	indomain_min	restart_constant	67028	3192	189	2	0,1584
input_order	indomain_min	restart_linear	5837	228	473	1	0,1648
input_order	indomain_min	restart_geometric	13352	650	4258	2	0,1611
input_order	indomain_min	restart_luby	2222	17	473	2	0,1782
input_order	indomain_random	restart_constant	53599	3154	3931	11	0,3755
input_order	indomain_random	restart_linear	26279	667	2854	9	0,2314
input_order	indomain_random	restart_geometric	59630	2994	14	11	0,2336
input_order	indomain_random	restart_luby	31052	1041	90	9	0,2746
first_fail	indomain_min	restart_constant	16654	784	1658	2	0,1737
first_fail	indomain_min	restart_linear	7305	288	486	2	0,1825
first_fail	indomain_min	restart_geometric	6642	288	346	2	0,1862
first_fail	indomain_min	restart_luby	10475	561	478	1	0,1671

Figure 2: Final model - instance no. 33

Final model with symmetry constraints - instance no. 18							
Variable heuristic	Domain heuristic	Restart heuristic	Propagations	Failures	Restarts	Solutions	Time (s)
input_order	indomain_min	restart_constant	406256825	24402840	928954	1	300,0000
input_order	indomain_min	restart_linear	27774568	1854274	37722	2	18,1191
input_order	indomain_min	restart_geometric	2349900	150432	23583	2	1,7518
input_order	indomain_min	restart_luby	7695844	602612	49415	2	5,6972
input_order	indomain_random	restart_constant	100343254	2969423	99284	12	57,1142
input_order	indomain_random	restart_linear	75913481	2393955	306388	8	42,0563
input_order	indomain_random	restart_geometric	25688757	916438	8093	13	14,6115
input_order	indomain_random	restart_luby	82965528	2657184	104208	13	49,2050
first_fail	indomain_min	restart_constant	422894583	25561280	933309	2	300,0000
first_fail	indomain_min	restart_linear	20181412	1256631	24198	2	23,6787
first_fail	indomain_min	restart_geometric	3773121	245736	8108	3	3,6035
first_fail	indomain_min	restart_luby	4750847	277346	43225	3	3,3549
first_fail	indomain_random	restart_constant	407885457	25741294	939594	10	300,0000
first_fail	indomain_random	restart_linear	25102363	767623	28647	11	29,9078
first_fail	indomain_random	restart_geometric	23678254	899253	94915	12	9,2888
first_fail	indomain_random	restart_luby	2159769	76212	5252	9	1,7923
dom_w_deg	indomain_min	restart_constant	407632118	24564868	1016317	1	300,0000
dom_w_deg	indomain_min	restart_linear	25431218	1631868	23601	2	16,6276
dom_w_deg	indomain_min	restart_geometric	3816303	250808	2436	3	3,0329
dom_w_deg	indomain_min	restart_luby	4731394	303641	55912	2	6,3359
dom_w_deg	indomain_random	restart_constant	408386840	25392823	1074713	11	300,0000
dom_w_deg	indomain_random	restart_linear	24974374	775884	32693	13	15,7069
dom_w_deg	indomain_random	restart_geometric	2904188	102095	5664	12	2,0477
dom_w_deg	indomain_random	restart_luby	16733193	520339	14847	14	9,8292

Figure 3: Final model with symmetry constraints - instance no. 18

Rotation model - instance no. 9							
Variable heuristic	Domain heuristic	Restart heuristic	Propagations	Failures	Restarts	Solutions	Time (s)
input_order	indomain_min	restart_constant	2512832508	8438410	4262973	1	300,0000
input_order	indomain_min	restart_linear	2819929963	10909615	101984	1	300,0000
input_order	indomain_min	restart_geometric	865939501	3155974	42171	1	90,6527
input_order	indomain_min	restart_luby	2695482978	9931654	2046459	1	300,0000
input_order	indomain_random	restart_constant	2737982321	9674743	3767973	6	300,0000
input_order	indomain_random	restart_linear	2750283462	10741027	108912	6	300,0000
input_order	indomain_random	restart_geometric	849415062	3086677	28041	6	91,4961
input_order	indomain_random	restart_luby	2789533308	10422297	1459403	6	300,0000
first_fail	indomain_min	restart_constant	2736384071	9157436	3765990	1	300,0000
first_fail	indomain_min	restart_linear	2851195497	11029427	220735	1	300,0000
first_fail	indomain_min	restart_geometric	823549674	3019363	43553	1	113,8587
first_fail	indomain_min	restart_luby	2836472276	10424713	2902882	1	300,0000
first_fail	indomain_random	restart_constant	2842320584	10045983	2641536	6	300,0000
first_fail	indomain_random	restart_linear	2860059241	11186186	184497	5	300,0000
first_fail	indomain_random	restart_geometric	849887922	3069934	1336	4	100,5887
first_fail	indomain_random	restart_luby	2909723532	10788655	1607576	5	300,0000
dom_w_deg	indomain_min	restart_constant	2688232504	9004506	4126812	1	300,0000
dom_w_deg	indomain_min	restart_linear	2875412793	11119188	141144	1	300,0000
dom_w_deg	indomain_min	restart_geometric	816716997	3063153	1691	1	80,5196
dom_w_deg	indomain_min	restart_luby	2827446195	10440668	2779816	1	300,0000
dom_w_deg	indomain_random	restart_constant	2783155964	9825471	2453202	6	300,0000
dom_w_deg	indomain_random	restart_linear	5928323839	12806264	149979	7	300,0000
dom_w_deg	indomain_random	restart_geometric	848651834	3152556	2058	5	98,9218
dom_w_deg	indomain_random	restart_luby	2644889173	9926216	1739399	5	300,0000

Figure 4: Rotation model - instance no. 9

Rotation model with symmetry constraints - instance no. 12							
Variable heuristic	Domain heuristic	Restart heuristic	Propagations	Failures	Restarts	Solutions	Time (s)
input_order	indomain_min	restart_constant	1231248	4562	942	2	0,2634
input_order	indomain_min	restart_linear	539746	1525	9677	2	0,2450
input_order	indomain_min	restart_geometric	5047135985	14198581	30384	1	300,0000
input_order	indomain_min	restart_luby	4661143392	15431065	1471961	1	300,0000
input_order	indomain_random	restart_constant	988704	3003	12077	9	0,2652
input_order	indomain_random	restart_linear	3707397519	14823971	255602	10	300,0000
input_order	indomain_random	restart_geometric	5032008135	16209018	47618	9	300,0000
input_order	indomain_random	restart_luby	10159915	21481	134772	10	0,9954
first_fail	indomain_min	restart_constant	1178279	5183	1934	2	0,2393
first_fail	indomain_min	restart_linear	2024103087	7071094	328657	1	300,0000
first_fail	indomain_min	restart_geometric	761350	2213	3145	2	0,2419
first_fail	indomain_min	restart_luby	2731475390	7716175	1478504	1	300,0000
first_fail	indomain_random	restart_constant	248306	428	179	11	0,2318
first_fail	indomain_random	restart_linear	150146	220	126	7	0,2033
first_fail	indomain_random	restart_geometric	888176	2320	15508	10	0,2631
first_fail	indomain_random	restart_luby	2468045660	7318913	1855403	10	300,0000
dom_w_deg	indomain_min	restart_constant	16988965	38016	1788	3	1,3012
dom_w_deg	indomain_min	restart_linear	4562038878	14952346	311758	1	300,0000
dom_w_deg	indomain_min	restart_geometric	5089948146	14895698	26087	1	300,0000
dom_w_deg	indomain_min	restart_luby	224618	637	685	2	0,1903
dom_w_deg	indomain_random	restart_constant	1230054	3663	463	9	0,3516
dom_w_deg	indomain_random	restart_linear	45320418	95436	81455	9	3,1290
dom_w_deg	indomain_random	restart_geometric	894509	3381	375	10	0,2897
dom_w_deg	indomain_random	restart_luby	3156227569	12095487	1350968	8	300,0000

Figure 5: Rotation model with symmetry constraints - instance no. 12

3.6.2 Time comparison

	Model			
Instance	Final	Final_symmetry	Rotation	Rotation_symmetry
1	0,172319	0,187567	0,148325	0,155662
2	0,137136	0,200555	0,140100	0,149530
3	0,137714	0,152447	0,149597	0,158366
4	0,146297	0,150504	0,174212	0,158838
5	0,131091	0,142577	0,390660	0,244757
6	0,140228	0,142032	0,742694	0,482241
7	0,127122	0,203774	0,768922	12,833886
8	0,166044	0,167699	0,165850	0,154067
9	0,187581	0,195296	121,967875	94,812512
10	0,318118	0,775979		
11	66,792261	238,538452		
12	0,588012	18,173632	0,988054	291,414720
13	0,336715	10,796957		
14	3,582063	15,578866		
15	2,728790	2,825569	2,264438	3,729467
16				
17	6,039608	112,484189	39,144670	72,369700
18	55,971575	291,039197		12,508238
19				
20	46,396877	67,384200		
21				
22				
23	62,092421	43,944552		3,892573
24	3,902522	103,225786	3,378998	3,587078
25				
26				
27	8,723359	8,638965	8,186404	9,070330
28	18,366714	15,165776		
29				
30				
31	74,532486	154,169594		44,739107
32				
33	188,911089	7,334389		6,546512
34				
35				
36	95,663876	12,846731		
37				
38				
39				
40				
Solved instances	**25**	**25**	**14**	**18**

Figure 6: Time comparison without heuristics

	Model (with best heuristics)			
Instance	Final	Final_symmetry	Rotation	Rotation_symmetry
1	0,1524	0,1225	0,1613	0,1504
2	0,2416	0,1732	0,1314	0,1307
3	0,2130	0,1261	0,1642	0,2018
4	0,1831	0,1317	0,2002	0,1776
5	0,1861	0,1301	2,9540	1,8569
6	0,1577	0,1464	2,1323	2,4607
7	0,1578	0,1663	53,3510	15,3454
8	0,1295	0,1366	0,1301	0,1324
9	0,1744	0,1667	84,1845	39,4086
10	0,1734	0,2004		
11	0,1852	122,9942		
12	0,2430	0,5346	0,1369	
13	0,1467	0,5577		
14	11,1258	1,3814		
15		1,2566	0,2338	0,1942
16		2,7130		
17		22,2120	0,1361	0,1401
18		1,0695	0,6096	
19				
20		253,6771		
21	0,2306	0,3711		
22				
23		0,1640		0,4281
24		0,2310	0,1386	0,1478
25				
26				
27		0,1504	0,1737	0,1409
28		0,2200		0,2161
29		0,2198	0,8052	0,6947
30				
31	1,6385	4,1318		17,6980
32	115,8445		0,5775	0,3730
33	0,1705	0,1439	0,2355	29,7426
34				
35		20,3756		
36	0,2994	61,4873		
37				
38				
39				
40				
Solved instances	19	29	18	19

Figure 7: Time comparison with heuristics

4 SMT

One of the main goal of SMT solver is that they can reason natively at higher level of abstraction, while still retaining the speed and automation of boolean engines. Whereas the language of SAT solvers is boolean logic, the language of SMT solvers is first-order logic. By allowing first order formalization, together with boolean and domain specific reasoning, they lead an inescapable little loss of efficiency. On the other hand, this provide a much more improved expressivity and scalability.

With the aim of making a SMT model, we used Z3, which is a state-of-the-art SMT solver from Microsoft Research. It integrates a host of theory solvers in an expressive and efficient combination.

4.1 Model

As we were satisfied with the structure of the CP model we decided to encode the SMT one in a similar way.

The coordinates of the circuits are encoded in two *IntVector* respectively called x and y. Then the width and the height of the circuit are in the arrays *dx* and *dy*. Next that we have defined the maximum plate height to minimize, under the name of *height* which is our objective function. The optimizer is given by *z3*.

4.2 Domain Reduction

In order to reduce the search space we have decided to reduce the domain of specific variables. The x value has to be greater than 0 and cannot assume a value greater than the width of the plate minus its width. Otherwise it wouldn't fit in the plate. It's possible to apply the same reasoning to the y value. We can express mathematically those constraints in the following way:

$$\bigwedge_{i \in C} x_i \geq 0 \land x_i \leq W - min(w)$$

$$\bigwedge_{i \in C} y_i \geq 0 \land y_i \leq \sum_{k \in C} h_k - min(h)$$

4.3 Main constraint

As global constraint we have disposed the *cumulative* ones in order to respect container boundaries and fill the contained avoiding holes. Moreover, to make sure no circuit would fall out the plate, we specified some boundaries constraint realized in the following way:

$$\bigwedge_{i \in C} x_i \geq 0 \land x_i + w_i \leq W - min(w)$$

$$\bigwedge_{i \in C} y_i \geq 0 \land y_i + h_i \leq \sum_{k \in C} h_k - min(h)$$

Those constraints tell the solver to verify that each coordinate plus its vertical or horizontal dimension respectively, is enclosed in the dimension of the plate.

Since the request of the problem, it's fundamental to dispose a constraint of not overlapping. This has been done translating in *z3* the following mathematical constraint:

$$\bigwedge_{i \in C, j \in C, i \neq j} x_i + w_i \leq x_j \lor x_j + w_j \leq x_i \lor y_i + h_i \leq y_i \lor y_i + h_j \leq y_i$$

4.4 Symmetry breaking constraint

With the aim to reduce the number of solutions the solver has to explore, we have defined symmetry breaking constraints. Like in CP, we decided to place always the biggest circuit in the bottom left part of the plate. Moreover the second one biggest has to be placed always on the right and/or on the top of the biggest one. The math used is the same defined before for symmetry breaking.

$$\forall i \in \{1..n\} : area_i = x_i * y_i$$

$$(y_{c1}, x_{c1}) \leq (y_{c2}, x_{c2}) \wedge x_{c1} * 2 \leq width \wedge y_{c1} * 2 \leq height$$

4.5 Rotation

With the intention of considering the possibility of rotation of each rectangle, it's necessary to bring some modification to the model. As before we considered the actual dimension of the plate verifying if the circuit is rotated. The right lengths are contained in two *IntVector*, respectively *dx_r* and *dy_r*.

$$\bigwedge_{i \in C} (dx\_r_i = dx_i \wedge dy\_r_i = d\_y_i) \vee (dx\_r_i = dy_i \wedge dy\_r_i = d\_r_i)$$

Since we allow the rotation of the rectangles, the solver will recognize as different two identical solution. As a matter of fact, for rectangles which are squares (same *width* and *height*) the solver will consider as different solution the two in which the square is rotated. Due to that, it's necessary to add another symmetry breaking constraint.

$$\bigwedge_{i \in C} dx_i = dy_i \implies (dx\_r_i = dx_i \wedge dy\_r_i = d\_y_i)$$

4.6 Results and performances

	Model			
Instance	Final	Final_symmetry	Rotation	Rotation_symmetry
1	0,018995	0,017168	0,104262	0,087597
2	0,034535	0,037667	0,227366	0,220544
3	0,057321	0,070948	0,236825	0,508854
4	0,212730	0,106387	1,902419	1,253705
5	0,294088	0,229361	7,271432	3,984140
6	0,428132	0,320045	6,053844	6,474458
7	0,497067	0,326703	9,821630	7,760952
8	0,416540	0,502032	5,513651	6,151402
9	0,627413	0,403914	30,429305	14,971004
10	2,509642	1,809711		
11				
12	5,091754	4,782482	262,194517	
13	4,460014	4,805040		
14	31,524302	8,604625		
15	9,213465	6,662895		
16				
17	36,123324	24,970414		
18	66,031913	40,576514		
19				
20				
21				
22				
23		109,421234		
24	60,300061	36,402128		
25				
26				
27		184,803282		
28	197,119170			
29				
30				
31	96,851865	8,512750		53,213281
32				
33	141,762018	13,164386		22,040691
34				
35				
36				
37				
38				
39				
40				
Solved instances	**20**	**21**	**10**	**11**

Figure 8: Time comparison with heuristics

5 Hardware specific

Our project was conducted on a machine with those specifications:

- Intel Core™ i5 dual-core (2,7 GHz)
- RAM DDR3 8 GB
- Intel Iris Graphics 6100 1536 MB
- MacOS Monterey 12.0.1

The version of the interpreter used is Python 3.8.3 with Minizinc 2.5.5.

6 Conclusion

In conclusion we have reached satisfying results both for CP and SMT. With CP we obtained the best performance with the final model with symmetry breaking constraints, in which we have reached 29 solved instances. In the statistics we have considered only optimal solutions, namely only whose which have been solved within 5 minutes.

References

[1] *The MiniZinc Handbook*. URL: https://www.minizinc.org/doc-2.3.0/en/.

[2] *Cumulative constraints*. URL: https://sofdem.github.io/gccat/gccat/Ccumulative.html#uid17828.