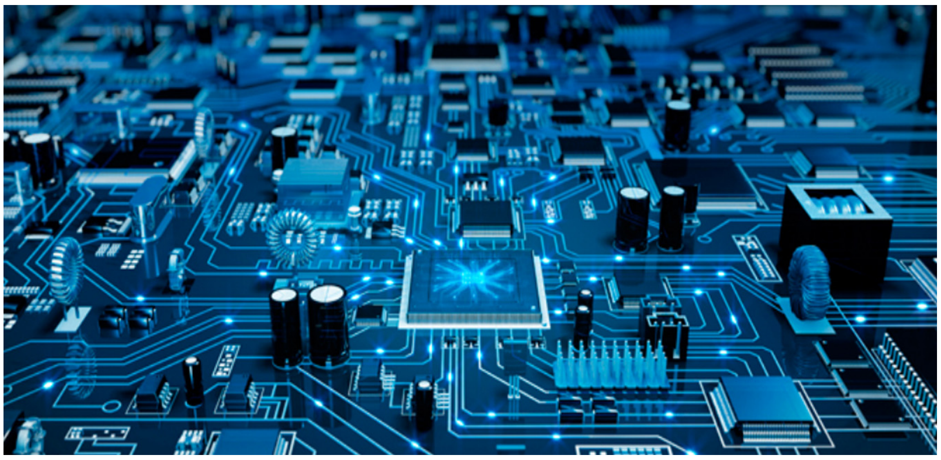
ALMA MATER STUDIORUM
UNIVERSITY OF BOLOGNA
School of Informatics, Science and Engineering

Master in

Artificial Intelligence

Combinatorial Decision Making and Optimization
Module 1
project work



MODELLING AND SOLVING THE VLSI PROBLEM

Student: **Bertè Mattia** ID: **ID 983469**

Accademic year 2020/2021

Contents

1 Description of the problem

VLSI (Very Large Scale Integration) refers to the trend of integrating circuits into silicon chips.
The VLSI can be seen as an application of the more general **strip packing problem**, known to be NP-Hard.
Given a set of axis-aligned rectangles and a strip of bounded width and infinite height, determine how to pack the rectangles into the strip minimizing its height and avoiding overlapping.
I faced the problem in two different ways, the first one using CP and the second one using SMT.

2 Modelling and solving with CP

First of all, from the data files, I have to load the parameters related to the fixed width of the chip W, the number of circuits n and height and width of each circuit.
I started with a basic modelling and then tried adding more constraints to exploit their propagation and to solve the problem more quickly.
Obviuosly the objective function is the height of the chip, and I want to minimize it:

$$\min_{c\ \in D} h(c) \tag{1}$$

where c are the circuits and D their set.

2.1 Basic constraint

The most basic constraints impose no overlapping between circuits and that circuits do not exceed the chip dimensions. The no overlapping constraint can be achieved with (2) and (3).

$$\forall i, j\ (x_i + w_i < x_j) \vee (x_j + w_j < x_i) \tag{2}$$

$$\forall i, j\ (y_i + h_i < y_j) \vee (y_j + h_j < y_i) \tag{3}$$

where x_i and y_i are the coordinates of the bottom left corner of the i-th circuit while w_i and y_i its dimensions.
It can be achieved in an easier way exploiting the built-in functions of MiniZinc, I can call the global constraint **diffn** that accept as input the coordinates and the sizes of the circuits.
Then I imposed the containment constraint:

$$\forall i\ (x_i + w_i \leq W) \wedge (y_i + h_i \leq H) \tag{4}$$

2.2 Implied constraint

As suggested in the point 2 of the assignment, I added two implied constraint (5) and (6). The sum of the widths along each row has to be smaller than W and the same for the sum of the heights along the column with respect to H.

$$\forall j = 1 \ldots H,\ \forall i\ s.t.\ y_i = j : \sum_i w_i \leq W \tag{5}$$

$$\forall j = 1 \ldots W,\ \forall i\ s.t.\ x_i = j : \sum_i h_i \leq H \tag{6}$$

Also these constraints can be achieved with the global constraint **cumulative**.

2.3 Symmetry breaking

In order to break the symmetry of the problem, I sorted the rectangles by their area in decreasing order. I imposed to the biggest rectangle coordinates the following constraint:

$$x_1 < 1 + \frac{(W - w_1)}{2} \tag{7}$$

$$y_1 < 1 + \frac{(H - h_1)}{2} \tag{8}$$

as suggested by Steinberg [1]. Doing so, I imposed to this circuit to be close to a vertex and it avoids the symmetrical solution that can be achieved

placing this circuit in the same position with respect to the other vertices.
Another symmetry breaking constraint can be achieved imposing a lexicographic order to the coordinates of circuits which have the same dimensions, doing so we avoid that the solver during the exploration try to swap these circuits leading to the same results.

$$\forall i = 1 \dots n-1,\ \forall j = i+1 \dots n,\ (w_i = w_j \wedge h_i = h_j) \implies lex([x_i, y_i], [x_j, y_j])$$

2.4 Additional constraints

In order to improve the model, I added some constraints to reduce the search space because they set boundaries for H.
The lower boundary can be computed as follow:

$$h_{min} = \max\left\{h_1, h_2, \dots, h_n, \frac{A}{W}\right\} \tag{9}$$

where A is the sum of the areas of each circuit 10.

$$A = \sum_{i=1}^{n} w_i \cdot h_i \tag{10}$$

The upper boundary can be proved to be:

$$h_{max} = 2 \cdot \max\left\{h_1, \dots, h_n, \frac{A}{W}\right\} = 2 \cdot h_{min} \tag{11}$$

as suggested by Steinberg.

2.5 Search strategy

With MiniZinc it's possible to select the way in which the search space is explored.
Since we are interested in minimizing the height, the idea is to try to assign before all the y_i, choosing before the variable with the smallest value in its domain and assigning it and then trying to assign a value to x_i starting with

the variable with the smallest domain size.

2.6 Results

I tried to solve each instance with both Chuffed and Geocode solver. The results are presented in Table 1. For both, I set a time limit of 5 minutes, if the solver requires more time it means that the specific instance can't be solved efficiently.
Obviuosly, when the optimal solution is found, it is equal for both the solver, indeed the whole search space has been explored and no better solution can be found. Other two possible results can be detected in the table:

- results marked with a star, they identify sub-optimal results. They represent the best solutions found within the time limit but since the search spaced hasn't been explored in its whole, we can't affirm for sure that they are the optimal one and usually they are not. Also sub-optimal solution are useful since they can be computed quickly and in some case they represent a good approximation of the optimal ones. The time column contains the time required to obtain the sub-optimal solution;

- No Sol means that no solution has been found within the time limits.

Regarding the time column, especially for the easiest problem, the time required is very low and it can vary, depending on the computational resources available, so each instance has been solved 5 times and it has been averaged. When the problem is easy the two solvers take similar time while with more complex problems the difference becomes more evident.

Instance	Circuits	Width	Geocode		Chuffed	
			Height	Time [s]	Height	Time [s]
1	4	8	8	0.52	8	0.55
2	5	9	9	0.54	9	0.51
3	6	10	10	0.62	10	0.52
4	7	11	11	0.53	11	0.54
5	8	12	12	0.54	12	0.53
6	9	13	13	0.54	13	0.52
7	9	14	14	0.63	14	0.66
8	10	15	15	0.52	15	0.57
9	10	16	16	0.55	16	0.51
10	12	17	17	0.64	17	0.63
11	16	18	18	1.22	No Sol	Time Out
12	14	19	19	0.59	19	0.53
13	14	20	20	0.62	20	0.61
14	15	21	21	0.69	21	0.67
15	16	22	22	0.58	22	0.52
16	19	23	23	283.73	25*	0.56*
17	18	24	24	0.73	24	1.12
18	19	25	25	0.71	25	0.56
19	22	26	26	2.12	26	6.08
20	21	27	27	0.97	27	0.89
21	22	28	28	1.32	28	36.48
22	24	29	29	1.64	29	27.79
23	20	30	30	0.97	30	7.25
24	19	31	31	0.68	31	0.56
25	27	32	32	1.83	32	23.12
26	23	33	33	7.32	35*	3.17*
27	21	34	34	22.24	36*	1.15*
28	22	35	35	0.76	35	0.59
29	23	36	36	292.86	37*	1.69*
30	27	37	37	0.89	37	0.76
31	19	38	41*	0.68*	41	0.56*
32	29	39	40*	4.21*	40*	2.85*
33	20	40	40	293.46	42*	0.54*
34	25	15	40	3.15	41*	0.56*
35	25	15	40	57.23	No Sol	Time Out
36	25	15	40	1.13	40	0.77
37	28	30	No Sol	Time Out	No Sol	Time Out
38	29	30	61*	5.43*	No Sol	Time Out
39	28	30	61*	1.01*	61*	4.12*
40	73	60	No Sol	Time Out	No Sol	Time Out

Table 1 - Table presenting the experimental results.

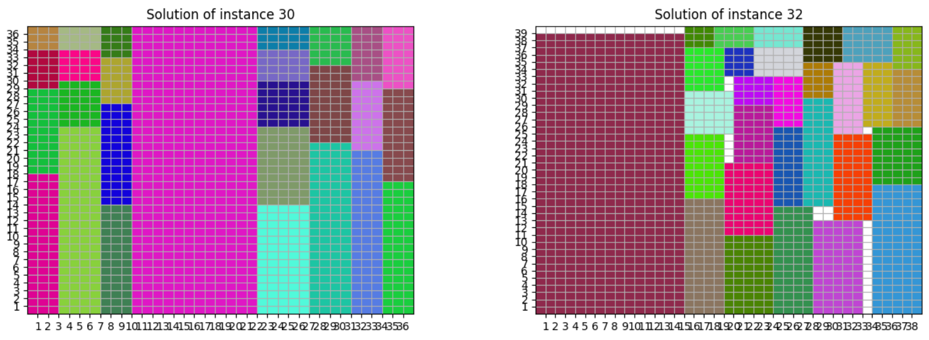


Figure 1 - Example of optimal *(Left)* and sub-optimal solution *(Right)*

In Figure 1, it's possible to see example of the solutions. When a solution is optimal is evident, indeed there is no empty space between the circuits and the height is equal to h_{min}.

2.7 Handling rotations

In the previous problems the circuits couldn't be rotated. In some practical applications this constraint is not present, so it can be useful to allow the rotation of circuits to see if better solutions can be obtained.
I added a binary variable r_i for each circuits and the following constraints:

$$\forall i \; w_i = (1 - r_i)d_{i1} + r_i d_{i2} \tag{12}$$

$$\forall i \; h_i = r_i d_{i1} + (1 - r_i)d_{i2} \tag{13}$$

where d_{i1} and d_{i2} define the dimensions obtained from the input file. So when $r_i = 0$, $w_i = d_{i1}$ and $h_i = d_{i2}$, while when $r_i = 1$, $w_i = d_{i2}$ and $h_i = d_{i1}$. It means that the circuit is rotated only when its correspondent variable r_i is equal to 1.
I added another simple symmetry breaking constraint:

$$\forall i \; (d_{i1} = d_{i2}) \implies r_i = 0 \tag{14}$$

it avoids to try to rotate square circuits. Another similar constraint is:

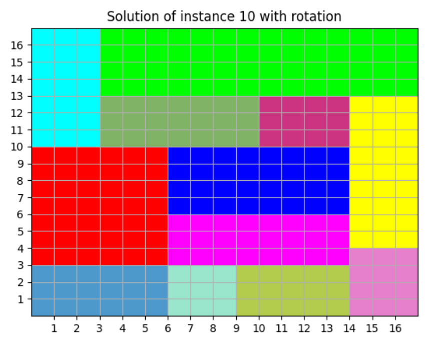
$$\forall i,\ \forall j\ (d_{i1} = d_{j2} \wedge d_{i2} = d_{j1}) \implies \neg(r_i = 1 \wedge r_j = 1) \tag{15}$$

when there are two circuits with the same dimensions values but in exchanged order, if both of them are rotated the solution is the same. So, constraint (15) avoids it.
As expected, allowing rotation enlarges the search space and consequently also the time required to explore it. In this case I used only Chuffed since it showed better performances and however the number of solved instances decreased a lot, in Table 2 are presented only the instances solved. We would need stronger symmetry breaking constraint to reduce time and search space.

Instance	Circuits	Width	Height	Time [s]
1	4	8	8	0.52
2	5	9	9	0.56
3	6	10	10	0.63
4	7	11	11	0.53
5	8	12	12	0.67
6	9	13	13	0.81
7	9	14	14	0.89
8	10	15	15	5.26
9	10	16	16	1.89
10	12	17	17	1.73

Instance	Circuits	Width	Height	Time [s]
11	16	18	18	64.32
12	14	19	19	9.88
13	14	20	20	38.27
16	19	23	24*	5.89
18	19	25	26*	153.38*
28	22	35	54*	130.20*
32	29	39	44*	139.9*
34	25	15	40	83.9
35	25	15	40	57.23
36	25	15	40	1.13

Table 2 - Tables presenting the experimental results.



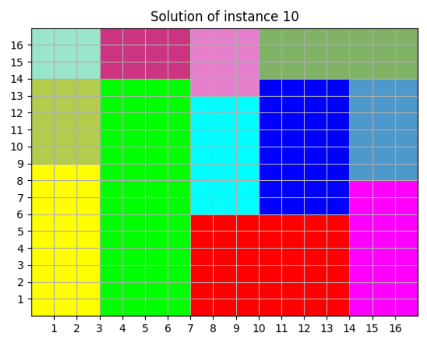


Figure 2 - Example of solutions of the same instance when rotation is allowed (Left) and when it's not.

3 SMT encoding and solving

I tried to encode the problem using the same constraints presented in the CP model. I discovered quickly that it doesn't work well as expected, indeed the solver was able to find solutions of just a few instances. Deleting the lexicographic constraint the solver started to produce solutions very quickly. One way to explain this is that the underlying DPLL algorithm keeps producing learned lemmas with the hope that it will find a contradiction with a previously known fact. The new constraints added might cause it to generate a lot of correct but irrelevant lemmas that makes it go down the deep-end with no useful result. Below the constraint used:

$$\forall i, j \ (x_i + w_i < x_j) \vee (x_j + w_j < x_i) \tag{16}$$

$$\forall i, j \ (y_i + h_i < y_j) \vee (y_j + h_j < y_i) \tag{17}$$

$$\forall i \ (x_i + w_i \leq W) \wedge (y_i + h_i \leq H) \tag{18}$$

$$\forall j = 1 \dots H, \ \forall i \ s.t. \ y_i = j : \sum_i w_i \leq W \tag{19}$$

$$\forall j = 1 \dots W, \ \forall i \ s.t. \ x_i = j : \sum_i h_i \leq H \tag{20}$$

$$x_1 < 1 + \frac{(W - w_1)}{2} \tag{21}$$

$$y_1 < 1 + \frac{(H - h_1)}{2} \tag{22}$$

$$h_{min} = \max\left\{h_1, h_2, \dots, h_n, \frac{A}{W}\right\} \tag{23}$$

$$h_{max} = 2 \cdot \max\left\{h_1, \dots, h_n, \frac{A}{W}\right\} = 2 \cdot h_{min} \tag{24}$$

In Table 3 are presented the optimal solution found, that correspond to those found by CP model, also in this case I solved each instance different times and then averaged them. In this case the variability of the time required was even grater due to the fact that it's not possible to determine the search strategy as in MiniZinc, each time infact the solution provided was different.

The results presented have to be considered as approximate.
The presence of the star near the solution indicates that the problem can't be solved always within the time limit.

Instance	Circuits	Width	Height	Time [s]
1	4	8	8	0.025
2	5	9	9	0.041
3	6	10	10	0.052
4	7	11	11	0.077
5	8	12	12	0.131
6	9	13	13	0.203
7	9	14	14	0.185
8	10	15	15	0.268
9	10	16	16	0.291
10	12	17	17	0.647
11	16	18	18	12.015
12	14	19	19	1.647
13	14	20	20	1.736
14	15	21	21	3.146

Instance	Circuits	Width	Height	Time [s]
15	16	22	22	2.931
16	19	23	23	33.642
17	18	24	24	10.567
18	19	25	25	13.638
19	22	26	26	270.218 ○
20	21	27	27	92.533
23	20	30	30	67.352
24	19	31	31	25.745
26	23	33	33	276.526 ○
27	21	34	34	57.625
28	22	35	35	87.327
29	23	36	36	224.371
31	19	38	40	27.135
33	20	40	40	26.732

Table 3 - Table presenting the experimental results.

3.1 Handling rotation

As in CP, I introduced the variables r_i to handle the rotation and then added the same constraint presented in section 2.7. As with MiniZinc, it requires more time and the number of solvable instances decreased (Table 4).

Instance	Circuits	Width	Height	Time [s]
1	4	8	8	0.061
2	5	9	9	0.078
3	6	10	10	0.136
4	7	11	11	0.183
5	8	12	12	0.859
6	9	13	13	0.972
7	9	14	14	1.164
8	10	15	15	0.455
9	10	16	16	0.629

Instance	Circuits	Width	Height	Time [s]
10	12	17	17	6.109
11	16	18	18	191.250 ○
12	14	19	19	60.652
13	14	20	20	44.133
14	15	21	21	73.697
15	16	22	22	96.338 ○
24	19	31	31	179.904 ○
31	19	38	40	138.561 ○
33	20	40	40	235,468 ○

Table 4 - Table presenting the experimental results.

4 Conclusions

In this section are presented some plot with the time required for each instance in order to make more explicit the comparison between different solver and techniques.
When an instance wasn't solvable, its time has been set equal to 500 s. Plots present the comparison between:

- Chuffed and Geocode time when rotation isn't allowed (Figure 3);
- time required by Chuffed to solve instances when rotation is allowed and when it is not (Figure 4);
- time required by Z3 to solve instances when rotation is allowed and when it is not (Figure 5);
- Chuffed and Z3 time when rotation isn't allowed (Figure 6);

As can be seen from the plot, for our specific problem and by the way I've modelled it, CP with Chuffed solver showed the best performances. However it doesn't mean that it is the best way to solver problem of this type. Probably with a deeper knowledge of the problem and of the combinatorial topic, we could end up with better modelling/encoding that allows to solve an higher number of instances and quicker.

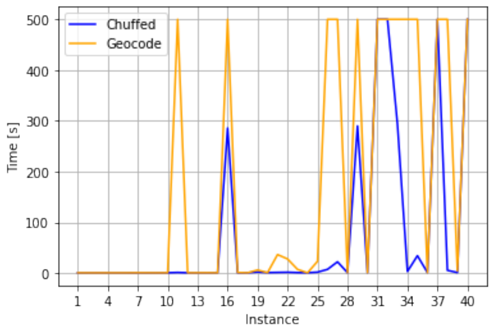


Figure 3

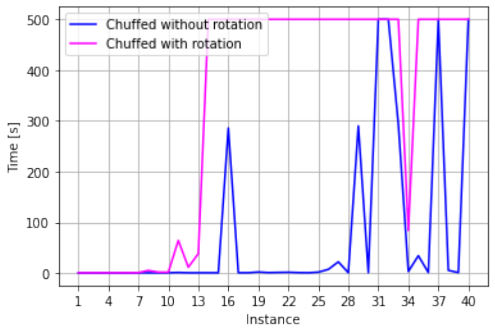


Figure 4

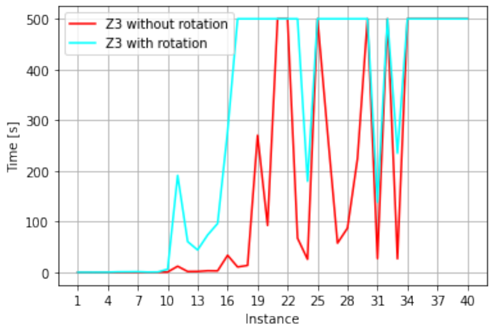


Figure 5

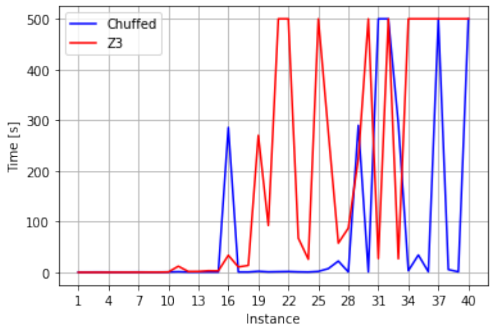


Figure 6

References

[1] A. Steinberg, “A strip-packing algorithm with absolute performance bound 2,” *SIAM J. Comput.*, vol. 26, pp. 401–409, 1997.