Giovanni Minelli

SAT and CP-based approach to VLSI problem

Project report for Combinatorial Decision Making and Optimization Module 1

University of Bologna
Academic Year 2020–2021

Abstract

Very Large Scale Integration (VLSI) 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 project aim is to face the problem with different approaches such as SAT, SMT and CP to evaluate their strenghts.

Introduction

Description of the problem

The problem require the design of the circuits defining your electrical device in a way to pack more transitors as possible into the same area of silicon. Formally, 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 (improving its portability). In order for the device to work properly, each circuit must be placed in a fixed orientation with respect to the others, therefore it cannot be rotated. Finally some hipothesis of implementation are advanced for a special case of the problem where rotations are allowed.
The problem is known in literature as 2 Dimension Strip Packing (2DSP) and is known to be NP-Hard. Moreover when rotations and translation are allowed, it is strongly NP-hard and it is even not known whether it is in NP, because it is complicated to encode rotations efficiently [1]

Following the specifications of the project the problem has been faced with different approaches, namely: CP, SAT and SMT.
To come up with the best model/encoding and tackle relatively more difficult instances of the problem, in the most efficient way, further analysis under different restriction has been taken in consideration. Also as mentioned by the specification of the assignment no assumptions based on the instances has been used.
Finally, for each approach, the results have been collected and presented in the relative section, to show with experimental data which is the best combination of techniques.

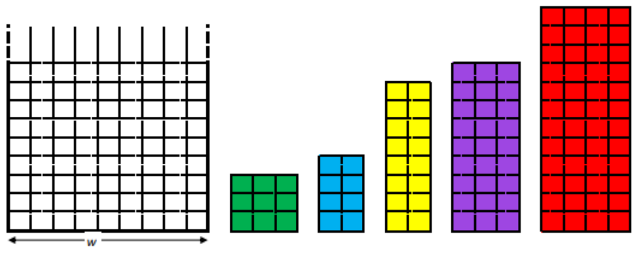


Figure 1: Graphical representation of the instance.

1 CP-based approach

1.1 Implementation

The implementation has been made with the python library of Minizinc able to inject the input parameters and execute the model. The python script is in charge of all the work of i/o handling and result presentation while the solution computation is performed by the Minizinc solver Gecode in the side script.

1.2 Step-by-step resolution

As suggested by the assignment paper our final implementation has been obtained following a step by step procedure, decoupling the details of the model for a finer analysis.

1.2.1 Variables, Main constraints and objective function

At first it's possible to see the definition of the input variables which values is read from the txt instance file and passed with the python script:

```
    % width of the circuit plate.
int: width;
    % number of circuits to be allocated in the instance
int: n_rects;
    % widths and heights of circuits
array[RECTS] of int: width_rects;
array[RECTS] of int: height_rects;
```

The definition of the output variables are:

```
    % circuits positions from bottom right corner
array[RECTS] of var 0..width : X;
array[RECTS] of var 0..max_height: Y;
    % bounding box height value
var min_height..max_height: HEIGHT;
```

and other useful variables:

```
    % index of the circuits
set of int: RECTS = 1..n_rects;
    % plate geometric bounds
int: min_height = max(height_rects);
int: max_height = sum(height_rects);
```

The main constraints that enable the resolution of the problem (possibly in an inefficent manner) are the simple displacement of the circuits inside the size bound of the plate and the non overlap constraint. An initial basic attempt is therefore like:

```
%equivalently for width and height
constraint
    forall(rect in RECTS)(
        X[rect] + width_rects[rect] <= width /\
        Y[rect] + height_rects[rect] <= HEIGHT
    );

constraint
    forall( m, n in RECTS where m < n ) (
        X[m] + width_rects[m] <= X[n]  \/
        X[n] + width_rects[n] <= X[m]  \/
        Y[m] + height_rects[m] <= Y[n] \/
        Y[n] + height_rects[n] <= Y[m]
    );
```

The objective function as stated by the problem, is to minimize the overall height of the circuit plate. In the MiniZinc program it has been represented with the variable `HEIGHT`.

1.2.2 Implied constraints

An implied constraint is a characteristic of the problem itself which is logically implied by structural properties but which cannot be deduced by the solver.
Even if redundant typically improves the propagation and permit to reduce the search space. In this case, we can model the constraint of allocated space for each circuit in a way it doesn't exceed the sizes of the plate as seen before.
The initial results with this basic model where very good in the small instances but the performances deteriorate rapidly with inputs of bigger size and in bigger number as can be observed in the result section.

Other redundant constraints tried but trashed since they weren't able to improve the performance of the solver where:

- Each circuit must have at least one side in common with another circuit.
- Each circuit must have one of its corners in touch with another circuit corner.

1.2.3 Global constraints

To improve the model, it has been evaluated the possibility of make use of some global constraints.

The plate size constraint just mentioned has been integrated with two cumulative constraints labeled as redundant to acknowledge the solver. Used for describing cumulative resources usage, it requires for example, that the set of circuits given by a vertical position in `Y`, height size in `height_rects`, and width requirement in `width_rects`, never require more than a global bound `width`.

```
constraint
    cumulative(Y, height_rects, width_rects, width) /\
    cumulative(X, width_rects, height_rects, HEIGHT);
```

Then another constraint for the circuit overlapping avoidance has been used: diffn. A global packing constraint which avoid overlaps simultaneously on both axes.

```
constraint diffn(X, Y, width_rects, height_rects);
```

Results obtained just with global constraints were very good returning an optimal value with most of the instances even if the number of failures is relatively high and in the remaining cases the computation time represented a bottleneck for the solution.

A final remark is needed to point out that in similar problem applications (2DBP with unloading constraint [2]) when additional optimizations, such as diffn and cumulative constraints are introduced, it was recorded a decrease in the model performance. That was attributed to unnecessary computation especially in case of small problems, therefore the slowdowns encountered are probably heavily dependent to the instances.

1.2.4 Symmetries

Symmetry-breaking can lead to the simplification and/or decomposition of complex global constraints, and therefore to a stronger final model. Possible symmetries discoverable in the search space are:

- equals circuits can be interchanged
- quadrilateral subregions fully occupied with circuits can allow internal rearrangements without affecting the shape of the region and so overall improvements
- also flip over the axis of the whole solution can be detected since won't lead to improvements

However weaker symmetry-breaking scheme might sometimes be preferred to a stronger one because of the implied constraints that can be derived from it [4]. For this understanding multiple combinations where evaluated aiming at the best model solver.

```
% symmetry breaking for equals circuits
constraint
  forall(r in RECTS)(
    let {array[int] of int: EQ = [i | i in RECTS where
width_rects[i]=width_rects[r] /\ height_rects[i]=height_rects[r]]} in
      if length(EQ)>1 /\ min(EQ)=r then
        forall(i in index_set(EQ) where i>1)(
          lex_less([Y[EQ[i1]],X[EQ[i1]]], [Y[EQ[i]],X[EQ[i]]])
        )
      else true endif
  );

% symmetry breaking for polygonal subregions of circuits interchangable
constraint let { var 0..n_rects-1: y_min; var 0..n_rects-1: x_min; var
1..n_rects: y_max; var 1..n_rects: x_max; array[2..n_rects-1] of var
1..n_rects: SUB} in
  if y_min>y_max /\ x_min>x_max /\ alldifferent(SUB) /\
  forall(rect in SUB)(
    X[rect] + width_rects[rect] <= x_max /\ x_min <= X[rect] /\
    Y[rect] + height_rects[rect] <= y_max /\ y_min <= Y[rect]
  ) /\
  forall(xi in  x_min..x_max)(
    exists(b in SUB)(X[b]<=xi /\ xi<=X[b]+width_rects[b] /\
y_max=Y[b]+height_rects[b]) /\
    exists(b in SUB)(X[b]<=xi /\ xi<=X[b]+width_rects[b] /\ y_min=Y[b])
  ) /\
  forall(yi in  y_min..y_max)(
    exists(b in SUB)(Y[b]<=yi /\ yi<=Y[b]+height_rects[b] /\
x_max=X[b]+width_rects[b]) /\
    exists(b in SUB)(Y[b]<=yi /\ yi<=Y[b]+height_rects[b] /\ x_min=X[b])
  ) then
    % set constraint to not allow other orderings when you have this
subregion of circuits
    forall(m,n in ordered_rects where m<n)(
      if(m in array2set(SUB) /\ n in array2set(SUB)) then
         lex_lesseq([X[m],Y[m]],[X[n],Y[n]])
       else true endif
     )
  else true endif;
```

The aforementioned events can generate multiple different solutions without possibility of improvement and at this regard these constraints aim to impose an order in a way to restrict the search space.
We can see the effects of symmetry breaking constraints in the tested instances, by a decrease in failures with a same time limit since pruning ability is augmented. Also, the results of testing showed a slight increment in the solving time with instances of bigger size wrt executions of the model without these additional constraints.
At last, the use of the final symmetry breaking constraint can be described as more harmful then else. In some test instances, it's presence leads to non optimal solutions or in some case no solution at all. Thinking about that a posteriori, and watching the plotted non optimal results, i blame the implementation which might be not properly correct. My hypothesis is that during the search, once found a valid subregion, the lex constraint is applied and maintained independently if the circuit's subregion is good and optimal or not. Maybe a more fine review of the code would lead to better results.
Detailed testing results of the best search model with and without symmetry breaking constraints can be found (for sake of brevity not included in the report) in the corresponding sources folder.

1.2.5 Search

For the search work, at first it has been observed how having the circuits ordered by size could have good impact on the resolution. For simplicity the input is sorted by area (w*h) in the python script. Anyway that step could be easily done in MiniZinc with `sort_by` function or by imposing an order constraint in a list:

```
constraint
    forall(m, n in RECTS where
width_rects[m]*height_rects[m]>width_rects[n]height_rects[n])(
        lex_greater([y[m],x[m]], [y[n],x[n]])
    );
```

This, in addition to the search strategy, allow to place at first the bigger circuits and improve the process. The search function in detail has been built performing a int search over the concatenation of the variables that we should valorize.
Finally the objective is a minimization of the `HEIGHT`

```
solve::
int_search([ HEIGHT ]
            ++ [ X[i] | i in RECTS ]
            ++ [ Y[i] | i in RECTS ],
            input_order, indomain_min, complete
            ) minimize HEIGHT;
```

Primary focus in the reasoning was given to the variable order: the size (HEIGHT) of the plate has been placed first in a way to discard as soon as possible all branches with too small value for that variable and then try to expand on the X axis before decide to displace vertically. Consistently with this idea `input_order` has been used as annotation for variable choice. For the strategy of choice of variable value assignment, the best results were obtained with `indomain_min` but also `indomain_split` works quite well leading to the optimal solution in most of the cases with same or less solving time but much more failures respect the other.
In the results section different search strategy were taken in consideration and also a comparison with a different variables order were tested. In particular, the results for that modified case weren't so distant to the optimal result obtained with precedence to horizontal axes. I think that such outcome is strongly dependent to the property of the single instance, and therefore a test instance can perform better with a certain order configuration having most of the circuits in input oriented in a certain direction.

1.2.6 Hypothetical model with rotation allowed

It's possible to consider an expanded model where the circuits to displace over the plate can also be rotated. A complete search would lead to a big increase of the time of executions since for each one the search process should consider two different versions. A very straightforward implementation would be the one of use the current model with the 2^n combinations of circuits, but that would be very inefficient. Instead, a more wise implementations would associate a richer domain to each circuit size dimensions. This would allow the possibility of use different pair of values (w and h) for it's displacement but still denying the possibility to use a single circuit more than once.

1.2.7 Hypothetical search with rotation allowed

The search process should probably take in consideration the orientation (which pair of sizes for each circuit) before the position of the circuit itself. To find those additional variable values, the search strategy for variable selection should prioritize minimum value of failures. That could be done with *dom_w_deg*, but an even better catch would be the use of *first_fail* as annotation. The choice of strategy for variable value selection should converge another time on *indomain_min* given the previous results,

1.3 Results

Even if the problem knowledge and the reasoning previously explained guided my primary decision of search strategy, many test were executed to evaluate different objective functions and search parameters for the search. In the latter case less relevant instances were not reported (namely 1 to 10) since most of the combinations performed well with very small effort.
Then, found the best search parameters and objective function, a full round of tests were conducted, confronting the results obtained with and without symmetry breaking and global constraints.

Here following the outcomes of test execute with a time bound of 300s for **height minimization**, order variables **H, Y, X**, search annotations **input_order, indomain_min, complete** :

	no sym and no global constraint			
N°	TIME (m)	SOLUTIONS	FAILURES	RESULT
1	00:00.97	1	16	OPT
2	00:00.52	1	1	OPT
3	00:00.52	1	54	OPT
4	00:03.80	1	1878046	OPT
5	00:00.52	2	34729	OPT
6	00:13.61	1	6972108	OPT
7	00:00.87	1	222086	OPT
8	00:00.86	1	224531	OPT
9	00:02.36	1	934767	OPT
10	05:00.22	0	107581821	_
11	05:00.24	0	72965224	_
12	05:00.23	0	97149669	_
13	05:00.24	0	105602893	_
14	05:00.26	0	102642636	_
15	05:00.23	0	88061187	_
16	05:00.25	0	50672387	_
17	05:00.26	0	59446286	_
18	05:00.61	0	67696063	_
19	05:00.26	0	46930499	_
20	05:00.27	0	67199703	_

	both sym				**only first sym**				**no sym constraint**			
N°	TIME	SOL	FAILS	RES	TIME	SOL	FAILS	RES	TIME	SOL	FAILS	RES
1	00:00	1	2	O	00:00	1	1	O	00:00	1	1	O
2	00:00	1	2	O	00:00	1	1	O	00:00	1	1	O
3	00:00	1	5	O	00:00	1	4	O	00:00	1	4	O
4	00:00	1	11	O	00:00	1	9	O	00:00	1	9	O
5	00:00	1	14	O	00:00	1	13	O	00:00	1	13	O
6	00:00	1	10	O	00:00	1	9	O	00:00	1	9	O
7	00:00	1	7	O	00:00	1	6	O	00:00	1	6	O
8	00:00	1	3	O	00:00	1	2	O	00:00	1	2	O
9	00:00	1	8	O	00:00	1	6	O	00:00	1	6	O
10	00:00	2	205	O	00:00	1	97	O	00:00	1	97	O
11	00:02	1	2155872	S	01:01	2	6691981	O	00:54	1	5614385	O
12	00:01	2	7387	O	00:00	2	6938	O	00:00	1	4288	O
13	00:00	1	10	O	00:00	1	8	O	00:00	1	8	O
14	00:01	2	1514	O	00:00	1	502	O	00:00	1	2003	O
15	00:00	1	5	O	00:00	1	3	O	00:00	1	3	O
16	00:02	1	3757	O	00:00	1	6537	O	00:00	1	6896	O
17	00:01	1	1679	O	00:00	1	4822	O	00:00	1	5425	O
18	00:02	1	3969	O	00:00	1	1601	O	00:00	1	2150	O
19	05:00	0	495887	_	03:48	1	14386971	O	03:02	1	11736168	O
20	01:07	1	131326	O	00:02	1	130463	O	00:01	1	43553	O
21	05:00	0	557969	_	00:22	1	1388863	O	00:15	1	958479	O
22	00:31	1	48631	O	00:01	1	51955	O	00:01	1	52476	O
23	00:01	1	33	O	00:00	1	18	O	00:00	1	18	O
24	00:01	1	2259	O	00:00	1	3381	O	00:00	1	3962	O
25	00:14	1	14194	O	00:00	1	5963	O	00:00	1	11445	O
26	05:00	0	476977	_	01:29	1	7037634	O	00:45	1	3797269	O
27	03:06	1	421871	O	00:05	1	490354	O	00:05	1	491290	O
28	05:00	0	624660	_	00:50	1	3410912	O	00:49	1	3349572	O
29	05:00	0	574666	_	00:29	1	2787024	O	00:01	1	52110	O
30	05:00	0	341755	_	05:00	0	26419253	_	05:00	0	24837618	_
31	01:03	1	253840	O	00:02	2	178058	O	00:06	2	670033	O
32	05:00	0	279973	_	00:04	1	450313	O	00:03	1	412280	O
33	05:00	0	987996	_	00:08	1	757723	O	00:58	1	6003368	O
34	05:00	0	488187	_	05:00	0	29280954	_	05:00	0	31704193	_
35	00:03	1	471231	S	00:16	1	1587583	O	00:15	1	1560246	O
36	00:02	1	1162	O	00:00	1	121	O	00:00	1	112	O
37	05:00	0	306468	_	00:24	1	1735449	O	00:23	1	1954095	O
38	05:00	0	277259	_	05:00	0	17270497	_	05:00	0	22002881	_
39	05:00	0	302736	_	05:00	0	19713445	_	05:00	0	21666919	_
40	05:00	0	300298	_	05:00	0	6806940	_	05:00	0	8602392	_

Table 1: The complete tests results can be found in the SAT sources folder

2 SAT-based approach

2.1 Implementation

This part of the project make use of of the Z3 theorem prover through the Python APIs: **Z3Py**. The previous script to read the instance and write the output solution has been reused. The solving part instead is now integrated in the same file but for conformity the name of input output variables were preserved.
Since my limited experience with SAT solvers in general I choose to solve the problem following an interpretation of the problem given by scientific literature, aiming for the optimality. name of input output variables were preserved.
The paper i relied on is [3], which not only explain the general approach used but also integrates many techniques in a way to reduce the search space.
From the experimental results section in the paper is also possible to see how that method was able to solve many problem instances including two open problems: their minimum heights are found and proved to be optimum.

2.2 Step-by-step resolution

The problem faced in the paper is a 2SPP (aka 2DSP - Two dimensional strip packing problem) which can be seen as an equivalent instance of the VLSI circuit problem of the assignment. The approach used apply a translation from it into a multiple resolution of a decision problem (2OPP) where the question to answer is "Does exist a displacement to pack the input rectangles in a strip of fixed height?" instead of querying what is the optimal height. Such idea was reproduced in the implementation but still in a step by step procedure to better evaluate the model built as suggested in the assignment.

2.2.1 Variables, Main constraints and objective function

For the formalization of the problem has been used the variables: *lr* and *ud*.
The used approach, instead of capturing the position of the circuits as in the CP-based section, make use of variables able to describe the space and inter-relations.
$lr_{i,j}$ is true if r_i are placed at the left to the r_j.
$ud_{i,j}$ is true if r_i are placed at the downward to the r_j.

Using a coordinate system starting from the bottom left corner, for each circuit i identified by the tuple $(x_i,\ y_i)$, the domain of positions in the plate is restricted to:

$$
\begin{aligned}
D(x_i) &= \{a \in \mathbb{N} \mid 0 \le a \le W - w_i\} \\
D(y_i) &= \{a \in \mathbb{N} \mid 0 \le a \le H - h_i\}
\end{aligned}
$$

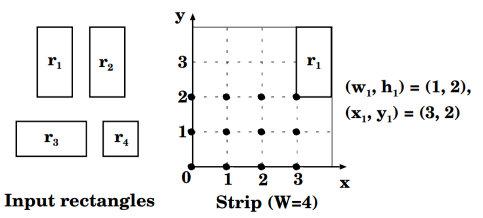


Figure 2: Domain representation of an example.

To constrain the solution to a feasible displacement we have to avoid the overlapping, and so for each rectangles r_i, r_j (i > j), the following constraints are used:

$$lr_{i,j} \vee lr_{j,i} \vee ud_{i,j} \vee ud_{j,i}$$
$$\neg lr_{i,j} \vee px_{i,e} \vee px_{j,e+w_i}$$
$$\neg lr_{j,i} \vee px_{j,e} \vee px_{i,e+w_j}$$
$$\neg ud_{i,j} \vee py_{i,f} \vee py_{j,f+h_i}$$
$$\neg ud_{j,i} \vee py_{j,f} \vee py_{i,f+h_j}$$

where the *px* and *py* represent a wise encoding for the displacement of a circuit (order encoding). In fact instead of capturing conflict points the constraints are encoded by representing conflict regions which lead to a large saving in the number of clauses used.
$px_{i,e}$ is true if r_i are placed at less than or equal to e.
$py_{i,f}$ is true if r_i are placed at less than or equal to f.

Then it's necessary a constraint for the ordering of such variables:

$$px_{i,e} \vee px_{i,e+1}$$
$$py_{i,f} \vee py_{j,f+1}$$

with e and f which take values in the domain space of each *i* circuit.

The suggested method of the paper to find the optimal value to answer the 2DSP problem is by iteratively call a 2OPP and make use of an additional variable *ph*.
ph_o is true if all rectangles are packed at the downward to the height *o*.
The following constraints allow to valorize it properly:

- $ph_o \vee py_{i,o-h_i}$ in case of existence of a valid displacement with fixed height *o* the *py* variable must be true for all circuits at the extent of the domain
- $ph_o \vee ph_{o+1}$ for order encoding

Since the 2OPP instances can span in the range *min_height* to *max_height* the optimal result can be found just iterating over the *ph* values, returning the minimum index *o* at which the variable is True.

2.2.2 Variables domain

From the structure of the problem itself we have that the feasibility of each circuit can be reduced operating on each domain. More precisely, each circuit can be displaced only at positions inside the plate keeping in consideration also it's sizes. Therefore, with a cycle over each circuit, the maximal extrema of the domain value are added as True clause for both px and py in case of a possible feasible displacement in the bounds of the plate. Instead, if the size is grater than a dimension of the plate, the same domain value is added to the solver with a False value causing the failure of the single decision problem.

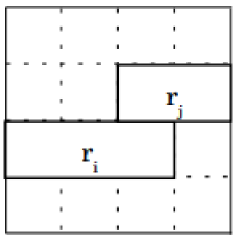
To mention something more in this regard, since the iterations for creation of the clauses could include literals also out of the domain allowed (e.g. $lr_{i,j} \vee px_{i,e} \vee px_{j,e+w_i}$), at first each domain included much more values than the necessary all of which had to be properly negated during the domain definition phase resulting in useless time spending.
To prevent that, the iterations for the clauses have been limited and also the resulting domain formulation for px and py variables become much more light. In practical terms the

solver statistics showed less memory usage and much less variables used.
Also, an important clause needed to reach a solution is the one which constraint the position of a rectangle j in case of $lr_{i,j}$ or $ud_{i,j}$ valorized to True: respectively $px_{j,wi-1}$ or $py_{j,hi-1}$ should be False

2.2.3 Symmetries

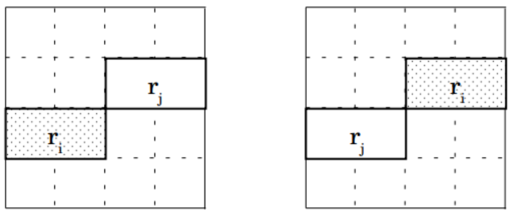
Different techniques to reduce the search space are evaluated considering symmetries and relations of rectangles.

- Large rectangles **(LR)** - Reducing the possibilities for placing large rectangles, both in horizontal and vertical direction. For each rectangle r_i and r_j, if $w_i + w_j > W$ we can not pack these rectangles in the horizontal direction, then we can eliminate useless overlapping constraints.



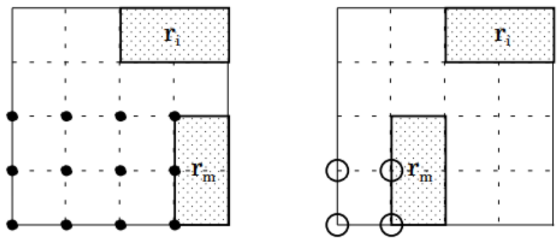
$$\cancel{lr_{i,j} \lor lr_{j,i}} \lor ud_{i,j} \lor ud_{j,i}$$
$$\cancel{\neg lr_{i,j} \lor px_{i,e} \lor px_{j,e+w_i}}$$
$$\cancel{\neg lr_{j,i} \lor px_{j,e} \lor px_{i,e+w_j}}$$
$$\neg ud_{i,j} \lor py_{i,f} \lor py_{j,f+h_i}$$
$$\neg ud_{j,i} \lor py_{j,f} \lor py_{i,f+h_j}$$

- Same rectangles **(SR)** - Breaking symmetries for same-sized rectangles. For each rectangle r_i and r_j, if $(w_i, h_i) = (w_j, h_j)$ we can fix the positional relation



$$lr_{i,j} \cancel{\lor lr_{j,i}} \lor ud_{i,j} \lor ud_{j,i}$$
$$\neg lr_{i,j} \lor px_{i,e} \lor px_{j,e+w_i}$$
$$\cancel{\neg lr_{j,i} \lor px_{j,e} \lor px_{i,e+w_j}}$$
$$\neg ud_{i,j} \lor py_{i,f} \lor py_{j,f+h_i}$$
$$\neg ud_{j,i} \lor py_{j,f} \lor py_{i,f+h_j}$$
$$\neg ud_{i,j} \lor lr_{j,i}$$

- Largest rectangle **(LS)** - Reducing the domain for the largest rectangle. We can force the position of the largest rectangle between a pair by removing some constraints. Allow the largest circuit to be displaced only at his maximal domain the other one will be affected as well with a domain reduction.



~~$lr_{i,j}$~~ $\vee lr_{j,i} \vee ud_{i,j} \vee ud_{j,i}$

~~$\neg lr_{i,j} \vee px_{i,e} \vee px_{j,e+w_i}$~~

$\neg lr_{j,i} \vee px_{j,e} \vee px_{i,e+w_j}$

$\neg ud_{i,j} \vee py_{i,f} \vee py_{j,f+h_i}$

$\neg ud_{j,i} \vee py_{j,f} \vee py_{i,f+h_j}$

2.2.4 Search

The search process have been the most hard detail to implement. The referenced paper assume an iterative approach of search for the optimal height, guided by SAT or UNSAT result of the many 2OPP (by use of a bisection method in the min/max height bounds).
My aim, following the requirements of the project assignment, was to make a single call to the solver with the unified clauses of each 2OPP. But, since the solver try to minimize the effort in the solution of the SAT problem the result obtained with the first encoding resulted always in the maximal height as solution and instead tightening the constraints resulted in an UNSAT result given by the unsatisfiability of the initials 2OPPs under optimal value. To overcome that, after many attempts, the *ph* variables have been modified to be always True except for the optimal height. Then iterating over the solution heights, that variable is put in relation with a property which have to be True in case of a proper height value able to satisfy the instance. Such case can indeed be verified checking the truthness of the extrema of the domain of *py* variable for each circuit (which size is tied to the size of the same circuit). Such clauses constraining the problem seemed correct with small instances but unfortunately at bigger ones the output solution wasn't optimal. Curiously in many cases the result was lower then the *max_height* value, and just reducing the space range of optimal height in which the solver has to search, the optimal value solution is always returned. In my opinion the bad encoding could have influenced the results given by the implementation.

2.2.5 Hypothetical model with rotation allowed

The extension of the model to include also rotations isn't faced in the paper but just cited as a future work extension.
A possible resolution method could be the one of consider the different combinations of circuits rotated, and execute separately the 2OPP on each one. Then, by using the cited bisection method forward the execution of only the branches which terminates with a SAT result for the current height value.

Implementing such relaxation in the model can instead be done encoding the circuit rotation with an hot encoded variable (a boolean for each rotation possible of each circuit) which value state about the rotation used for the solution and then during the clause creation could be considered interchangeable the value of height and width (such clause must relate such choice to the truthness of one rotation variable).

2.3 Results

The experimental results were collected considering the backjumps of the solver, the optimality of the solution and time of solving used by the SAT solver. Anyway just the instances from 1 to 16 were evaluated for each model combination since the execution time of the python script itself for the solver construction (formulating and adding the clauses) become very long without returning good results: the 16th instance exceeded 20 minutes of computation. Up to the eleventh the whole script execution last less then 300 seconds. The data collected were primary aimed to confront the different techniques (here showed just the best symmetry combination) and also it's possible to see how a different ordering can affect the failures/backjumps of the solver process but not the time of solving or the results obtained, differently from the CP-based approach.

	ORDERED			SYM ORDERED (best LRH+SR)		
N°	TIME (s)	BACKJUMPS	OPT	TIME (s)	BACKJUMPS	OPT
1	0,02	5	Y	0,02	3	Y
2	0,05	1	Y	0,05	1	Y
3	0,10	10	Y	0,10	8	Y
4	0,16	93	Y	0,14	37	N
5	0,58	227	N	0,66	229	N
6	0,89	327	N	0,94	283	N
7	1,39	193	N	1,40	256	N
8	2,03	29	Y	2,11	162	Y
9	2,37	151	Y	2,43	137	Y
10	4,51	869	N	4,65	103	N
11	12,56	327	N	10,13	434	N
12	14,29	148	N	13,70	148	N
13	18,61	319	N	18,07	319	N
14	22	304	N	21,10	304	N
15	29,83	263	N	29,72	1611	N
16	56,59	2230	N	56,52	2230	N

Table 2: The complete tests results can be found in the SAT sources folder

3 SMT-based approach

This section show the resolution of the given problem by using a SMT solver. Satisfiability Modulo Theories (SMT) solvers takes systems in arbitrary format (first-order logic), while SAT solvers are limited to Boolean equations and variables, nevertheless they still mantain the speed and automation of today's Boolean engines.

3.1 Implementation

For the implementation of such part it has been used the Z3 Solver's Python API: **Z3Py**. Most of the script used in SAT was reused since the problem and the behaviour of i/o of the program didn't change.
This implementation make use, as before, of the approach reported in [3]. A different resolution approach could have made use of the knowledge acquired in the construction of the CP model but i preferred to face the problem with the suggestions of the paper used in the SAT part since the symmetries and other techniques to reduce the search space seemed very reliable despite the mediocre result obtained in the previous implementation.

3.1.1 Variables, Main constraints and objective function

The original paper, as described in the chapter before, covered only a SAT-based approach and therefore a bit of work of conversion was needed. Fortunately the expressive power of a SMT solver is higher then a SAT solver and so the implementation of the problem resulted in a much more cleaner piece of code. In particular making use of the optimization ability of the Z3 library, the previous work aimed to unify different decision problems executed with a different height is no more needed.

Partially similarly to the SAT implementation it has been used the boolean variables: *lr* and *ud*.
$lr_{i,j}$ is true if r_i are placed at the left to the r_j.
$ud_{i,j}$ is true if r_i are placed at the downward to the r_j.

To represent the position in the space of each circuit, differently from before, a normal encoding has been used resulting in the integer variables X and Y, with the domain properly constrained as follow:

$$0 \le x_i \le W - w_i$$
$$0 \le y_i \le H - h_i$$

Where W, H are the sizes of the plate and w_i, h_i the dimensions of circuit i.
Also the domain of H is constrained between *min_height* and *max_height* valorized respectively as the maximum height of the circuits in input and the sum of all the circuits heights.

To constrain the solution to a feasible displacement we have to avoid the overlapping, and so for each rectangle r_i, r_j (i > j), the following constraints are used:

$$\begin{gathered} lr_{i,j} \lor lr_{j,i} \lor ud_{i,j} \lor ud_{j,i} \\ \neg lr_{i,j} \lor x_i + w_i \le x_j \\ \neg lr_{j,i} \lor x_j + w_j \le x_i \\ \neg ud_{i,j} \lor y_i + h_i \le y_j \\ \neg ud_{j,i} \lor y_j + h_j \le y_i \end{gathered}$$

Equivalently as before, just with the difference that the px and py variables have been substituted adapting the constraints to the new encoding with X and Y.
All other constraints become unnecessary and also the objective function can be easily expressed by telling the solver (optimizer) that the solution must minimize the value of H.

3.1.2 Symmetries

All techniques of search space reduction (except for the Large rectangles (**LR**) in vertical direction) as described before, has been implemented also for this approach. The difference from the previous part is just the definition of the constraints since the different spatial encoding.

3.1.3 Hypothetical model with rotation allowed

The encoding process similarly as before, can make use of the one hot encoding technique applied to a variable which indicate the rotation. Then the creation of additional clauses enable the other displacement, all tied to the truth value of the corresponding boolean variable which state the unique rotation for the circuit.

3.2 Results

To evaluate the results has been used the number of conflicts in the optimization process, the time of execution and the value of optimality in output. To be noted that in this case as value of time has been used the one of entire execution of the python script since it was just slightly greater than the real time measured for the execution of the optimizer, anyway the bound of 300s was maintained.
Confronting the results obtained with different combinations of symmetry breaking constraints it's possible to see that there is a decremental ordering by performance between models with no constraints at all, and models with all constraints enabled. This finding in the data collected, is valid both for the time and the number of conflict, but not for the optimality measure.
The best results are obtained with the model `SR+LRH` and `SR` which are able to solve 3 more instances in the time limit wrt the worst one. Anyway, even if both obtained similar results in most of the cases, for some instances in which both fail, the `SR` model report less conflicts.
Finally, as for the other approaches a comparison of execution with non ordered inputs have been made: the best model and the worst where tested. The results varied a lot for all the measure, in some cases the time and the conflicts decreased noticeably in one direction and sometimes increased. Also for what concern the optimality, the best model with unordered data solved three instances less than is counterpart, instead the worst model performed better with unordered data solving two more instances. Such result can be surely justified by a lucky occasional sequence and it's possible to conclude that both the SMT and SAT solver are randomly influenced by the presence of an ordering in the input data.

In conclusion, as it can be observed, the results obtained by the SMT solver are much better in comparison with the ones obtained by the SAT model. I can't exclude errors in my implementations however it's easy to notice how the encoding of the problem become easier when there isn't the boolean logic limitation.
To be noted also the gap of performance between the CP model and the other two.

	SR + LRH + LS (worst)			SR (best)			NO SYM OPT (ref)		
N°	TIME (s)	CONF	OPT	TIME (s)	CONF	OPT	TIME (s)	CONF	OPT
1	0,02	55	Y	0,02	33	Y	0,02	33	Y
2	0,03	29	Y	0,03	15	Y	0,03	15	Y
3	0,06	237	Y	0,05	250	Y	0,05	250	Y
4	0,07	364	Y	0,11	488	Y	0,09	488	Y
5	0,14	1263	Y	0,13	1130	Y	0,13	1130	Y
6	0,24	1779	Y	0,24	1879	Y	0,24	1879	Y
7	0,2	1415	Y	0,19	1309	Y	0,19	1309	Y
8	0,27	1101	Y	0,2	682	Y	0,2	682	Y
9	0,22	1080	Y	0,31	2456	Y	0,32	2456	Y
10	0,75	3736	Y	0,75	4016	Y	0,8	4016	Y
11	300,2	728922	N	300,21	582553	N	300,21	641937	N
12	2,35	10366	Y	1,31	5245	Y	2,64	12102	Y
13	1,99	18879	Y	3,03	29230	Y	1,76	18099	Y
14	7,8	41952	Y	4,17	30947	Y	12,14	67300	Y
15	6,33	18293	Y	4,51	13814	Y	2,34	6209	Y
16	300,35	161395	N	300,37	145302	N	300,31	725348	N
17	14,08	23609	Y	43,48	52332	Y	41,98	47220	Y
18	61,03	64376	Y	16,07	22683	Y	29,15	37307	Y
19	300,48	173884	N	300,47	123977	N	300,46	352363	N
20	300,43	515915	N	300,43	280142	N	300,44	217074	N
21	300,64	137136	N	300,44	373762	N	300,46	171396	N
22	300,57	230417	N	300,57	134636	N	300,62	170341	N
23	132,16	82176	Y	195,18	103645	Y	74,09	52302	Y
24	29,78	31116	Y	35,32	35121	Y	22,59	29740	Y
25	300,71	411406	N	300,75	90468	N	300,76	154342	N
26	300,69	141064	N	300,53	131410	N	300,53	139783	N
27	300,46	118982	N	122,17	71864	Y	166,58	84811	Y
28	300,5	119353	N	79,82	49680	Y	123,19	67280	Y
29	300,63	111471	N	294,88	106741	Y	300,53	112203	N
30	300,77	61330	N	300,85	61614	N	300,78	66306	N
31	28,68	36115	Y	25,82	28035	Y	37,85	40026	Y
32	300,86	47255	N	300,83	74715	N	300,84	60279	N
33	16,99	25312	Y	28,63	29703	Y	23,63	28850	Y
34	300,59	138649	N	300,57	147151	N	300,57	160877	N
35	300,71	136249	N	300,61	134957	N	300,7	121599	N
36	300,58	151275	N	300,6	172050	N	300,75	138121	N
37	300,69	110861	N	300,76	102092	N	300,87	105005	N
38	300,97	96099	N	300,79	95413	N	300,99	80875	N
39	300,97	124523	N	300,77	106013	N	300,82	129613	N
40	304,35	16514	N	304,19	18152	N	304,48	16381	N

Table 3: The complete tests results can be found in the SMT sources folder

References

[1] `http://courses.csail.mit.edu/6.890/fall14/scribe/lec2.pdf`

[2] `https://www.sciencedirect.com/science/article/pii/S0305054812002353`

[3] A SAT-based Method for Solving the Two-dimensional Strip Packing problem
`http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.163.7772&rep=rep1&type=pdf`

[4] Symmetry-breaking as a Prelude to Implied Constraints: Constraint Modelling Pattern
`https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.3.1051&rep=rep1&type=pdf`