Giovanni Minelli

SAT and CP-based approach to VLSI problem

Project report for Combinatorial Decision Making and Optimization Module 1

giovanni.minelli2@studio.unibo.it

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 ineffcient 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 [1]) 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 [2]. 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

References

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

[2] 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