VLSI Report

Giacomo Zamprogno, giacomo.zamprogno@studio.unibo.it
Carlo Cena, carlo.cena@studio.unibo.it

July 2021

Contents

1 Constraint Programming **2**
1.1 Problem Description, main constraints and objective function . . 2
1.2 Best Model 2
1.2.1 Horizontal and Vertical sum 2
1.2.2 Global constraints 3
1.2.3 Symmetry breaking constraint 3
1.3 Rotations 4
1.4 Search & Results 4

2 SAT **5**
2.1 Problem description and main constraints 5
2.2 Encoding 5
2.3 Model in direct encoding 6
2.4 Model in order encoding 6
2.5 Optimization 7
2.5.1 Implicit domain limitations 7
2.5.2 Implicit positional relation constraint 8
2.5.3 Domain reduction of largest rectangle 8
2.5.4 Positions of large dimension rectangles 8
2.5.5 Equal rectangles 8
2.6 Bisection algorithm for H_{min} 8
2.7 Results 9
2.8 Suggestions for including rotation 10

3 SMT **11**
3.1 Model 11
3.2 Results 12

1 Constraint Programming

The following paragraphs show how the given problem was encoded in MiniZinc and the results obtained through these encodings.

1.1 Problem Description, main constraints and objective function

The code is available in "model_0.mzn", the constraints used are those that describe the problem; given a 2D-array, called "dims", that represent the width and the length of each circuit and another 2D-array, called "sol", that represent the position of left-bottom corner of each circuit, and "w" the width of the enclosing rectangle, we have:

1. The variables of "sol" are in the range $(0, \max(max\_width, max\_height))$, where:
   (a) $max\_width = \min(w, sum(c \text{ in } CIRCUITS)(dims[c, 1]))$
   (b) $max\_height = 2 * \max(\max(dim\_ver), min\_height)$, property shown by [3]
   (c) $min\_height = floor(sum(c \text{ in } CIRCUITS)(dims[c, 1]*dims[c, 2])/w)$
2. The objective function, obj to be minimized is the maximum of the array "heights", with $heights[i] = sol[i, 2] + dims[i, 2]$;
3. obj, is such that $min\_height \leq obj \leq max\_height$;
4. For each pair of circuits (c1, c2) the following constraint needs to hold, in order to avoid intersections: $sol[c1, 1]+dims[c1, 1] \leq sol[c2, 1] \vee sol[c2, 1]+dims[c2, 1] \leq sol[c1, 1] \vee sol[c1, 2] + dims[c1, 2] \leq sol[c2, 2] \vee sol[c2, 2] + dims[c2, 2] \leq sol[c1, 2$;
5. Each circuit c needs to stay inside the given width, so $sol[c, 1]+dims[c, 1] \leq w$.

The time/instance plot for this model is shown in figure 1.

1.2 Best Model

The code is available in "model_2.mzn", the paragraphs below describe the encodings of all the suggestions available on the project description.

1.2.1 Horizontal and Vertical sum

We know that for each circuit, c, $sol[c, 1]$ should be between 0 and the maximum width, showed above, minus the length of the circuit, dims[c, 1]; the same can be said for the vertical coordinate, sol[c, 2], which should be between 0 and the maximum height, described in the above paragraph, minus its height, so to recap:

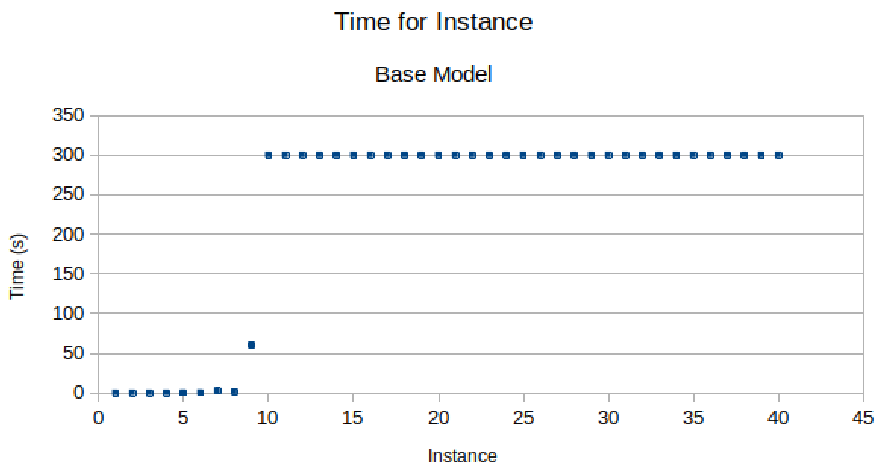


Figure 1: Base Model for cp, time required to solve each instance.

1. constraint forall(c in $CIRCUITS$)($sol\_tmp[c, 1] \leq max\_width - dims[per[c], 1]$)
2. constraint forall(c in $CIRCUITS$)($sol\_tmp[c, 2] \leq max\_height - dims[per[c], 2]$)

1.2.2 Global constraints

One can rewrite the constraints used to avoid intersection using the global constraint "diffn", moreover two constraints could be added using "cumulative" both on the horizontal and vertical axis to improve propagation, as shown below:

1. constraint diffn($sol\_hor, sol\_ver, dim\_hor, dim\_ver$)
2. constraint cumulative($sol\_ver, dim\_ver, dim\_hor, max\_width$)
3. constraint cumulative($sol\_hor, dim\_hor, dim\_ver, obj$)

1.2.3 Symmetry breaking constraint

This problem has various symmetries, each circuit disposition could be flipped on the x or y axis and the position of each circuit with equal shape in one or both axis could be changed with that of similar circuits; in order to restrict the search space the following symmetry constraints have been used:

1. Lexicographic constraints on both axis for circuits with equal shapes;
2. For each pair of circuits with only an equal dimension: $sol[c1, x] == sol[c2, x] \Rightarrow sol[c1, y] \leq sol[c2, y]$;

3. The biggest circuit is fixed in the left-bottom quadrant of the enclosing rectangle, i.e. constraint $sol\_tmp[1,1] \leq 1 + (w - dims[per[1],1])/2 \ \wedge$ $sol\_tmp[1,2] \leq 1 + (max\_height - dims[per[1],2])/2$, where per is a 2D-array that is used to keep track of the decreasing order of the circuits' areas.

1.3 Rotations

The model is provided in "model_rotation.mzn"; for each circuit the model can choose between the original and the rotated one, in order to support this choice a few additional structures have been created, such as:

1. 2D-array dims_r, for dimensions of both rotated and original circuits;
2. 2D-array sol_r, xy positions of left-bottom vertex for each circuit, for both rotated and original version;
3. 1D-array "taken", where taken[i] says if circuit i is rotated or original.

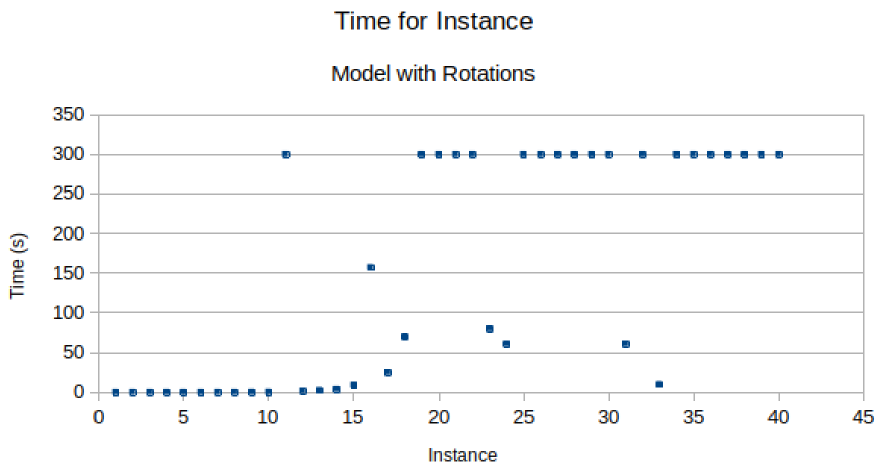


Figure 2: Model with rotations for cp, time required to solve each instance.

1.4 Search & Results

The best results have been obtained by sorting the circuits, based on their areas, from biggest to smallest and then searching with restarts, input order, random domains and gecode as solver.
In figure 3 is shown a plot of the time required to solve each instance with the best model, while in figure 2 is shown the plot for the model with rotations.

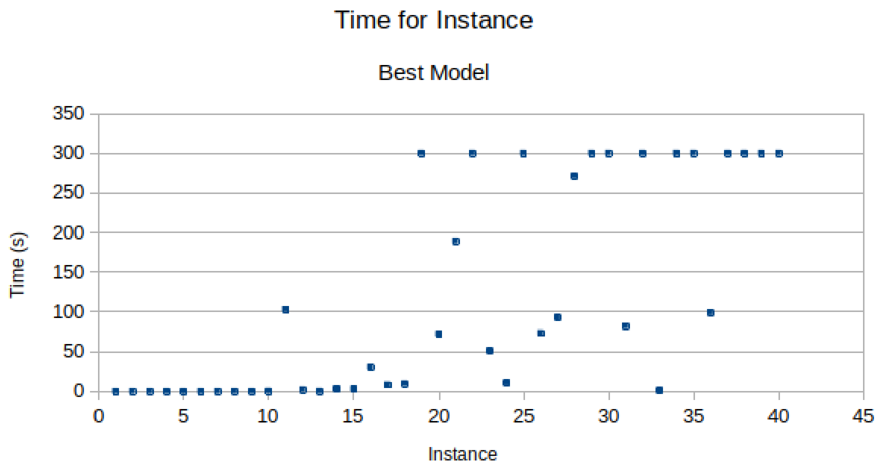


Figure 3: Best model for cp, time required to solve each instance.

2 SAT

2.1 Problem description and main constraints

The code is available in the notebook sat.ipynb and consists of three different models, the best of which is *solver(rects, W, opt=True, order=True)*. The main differences between the models, better detailed on the following sections, consist in the type of encoding used and the optimization steps.

In general, the problem at hand is based on 4 type of literals: literals of the type *px* and *py*, which encode the values of the coordinates of the rectangles, and literals of the type *left* and *under*, which encode the relative positions among the rectangles.

The fundamental constraint is so expressed:for each pair of rectangles, given a suitable implementation of the meaning for *left* and *under*, it must hold that:

$$r_i\_leftof\_rj \vee r_j\_leftof\_r_i \vee r_i\_under\_r_j \vee r_j\_under\_r_i \tag{1}$$

2.2 Encoding

Two approaches to encodings were taken: direct encoding and order encoding. They both share the same boolean variables, but their meaning and interpretation is different.

The literals encoding the coordinates of the rectangles are of the type px_i_X and py_i_Y.

In direct encoding, they have the intuitive meaning of ”the lower-left corner of the i-th rectangle has coordinates x=X and y=Y”. This encoding is the most

straightforward, but quickly becomes relatively intractable for larger scale problems due to the fast increase in the number of clauses required to cover all the possibilities.
The usage of order encoding was suggested in a paper by *Soh et al*[2], and it consists in interpreting the literals as "the lower-left corner of the i-th rectangle has coordinates $x \leq X$ and $y \leq Y$". This encoding is particularly suited for this kind of problem, since most of the constraints can be seen as "order" relations, and indeed it requires a lower number of statements, becoming more scalable even for smaller model. On the other side, the encoding itself is less intuitive and straightforward.

2.3 Model in direct encoding

Due to the way in which the literals in direct encoding directly represent the value of the coordinates, they must be mutually exclusive. For each rectangle r, one and only one of each px_r_X must be verified, and the same must hold for py_r_Y, this is performed by the functions $at\_least\_one(v)$, which adds a clause composed by the "or" of each interested literal, and $at\_most\_one(v)$, which for each couple of literals l1,l2 adds $\neg(l1 \wedge l2)$.

The meaning of *left under* can be seen, in higher logical language as:

$$r_i\_leftof\_rj \Rightarrow x_i + w_i \leq x_j \tag{2}$$

$$r_i\_under\_rj \Rightarrow y_i + h_i \leq y_j \tag{3}$$

Its translation in direct encoding requires various clauses: for each couple of permutation of rectangles ri and rj, and for each value x in $(0..wi - 1)$,

$$\neg r_i\_leftof\_r_j \vee \neg px_j\_x \tag{4}$$

for each value x' in $(W - w_i - w_j..W)$,

$$\neg r_i\_leftof\_r_j \vee \neg px_i\_x' \tag{5}$$

and for each value xi in $(0..W - w_i - w_j)$ and xj in $(0, x_i + w_i - 1)$,

$$\neg r_i\_leftof\_r_j \vee (\neg px_i\_xi \vee \neg px_j\_xj) \tag{6}$$

The same clauses, changed in accordance to the values, are included in order to describe *under*.

2.4 Model in order encoding

As it can be imagined by the name, order encoding forces an order relation between its variables. As a consequence, they are not mutually exclusive and instead, given a true literal px_i_X, all the literals of the type px_i_X', with

$X' > X$ will be true as well (same hold for y): for each rectangle r and for each value e in $(0..W-1)$ and f in $(0..H-1)$ the following clauses will be added

$$\neg px_i\_e \vee px_i\_(e+1) \tag{7}$$

$$\neg py_i\_f \vee py_i\_(f+1) Whichcanbebetterseenasseriesofimplicationsconstrainingthepossibilityoftherelativeposi \tag{8}$$

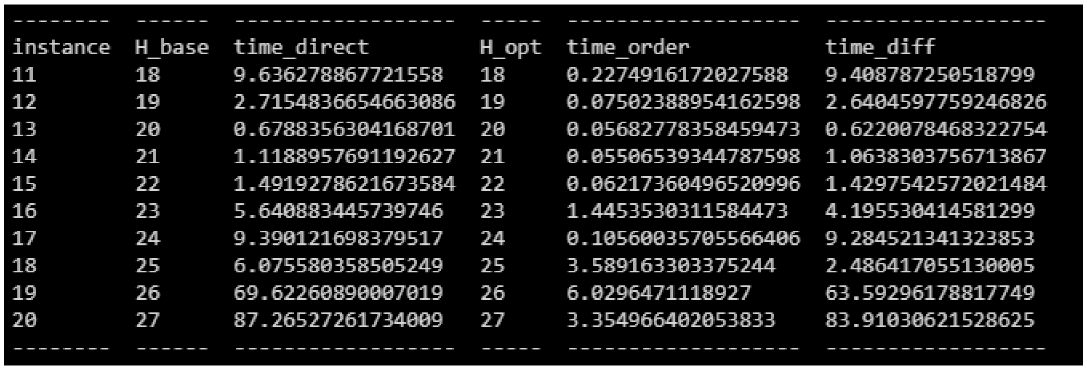
Which can be seen as a series of implication for which each true $\leq$ relation implies the same to hold for each greater value. In order encoding, the meaning of left and under translates into: for each possible permutation of rectangles ri and rj, and for each e in $(0..W-w_i)$, and f in $(0..H-h_i)$ it holds:

$$\neg r_i\_leftof\_rj \vee px_i\_e \vee \neg px_j\_(e+w_i) \tag{9}$$

$$\neg r_i\_under\_rj \vee py_i\_f \vee \neg py_j\_(f+h_i) \tag{10}$$

2.5 Optimization

After the first tests between the two types of encoding, it was clear that the model in order encoding was outperforming the one in direct encoding, which was as a consequence dropped. The following steps were performed in order to further improve performance for order encoding:



```
--------  ------  ------------------  -----  -------------------  ------------------
instance  H_base  time_direct         H_opt  time_order           time_diff
11        18      9.636278867721558   18     0.2274916172027588   9.408787250518799
12        19      2.7154836654663086  19     0.07502388954162598  2.6404597759246826
13        20      0.6788356304168701  20     0.05682778358459473  0.6220078468322754
14        21      1.1188957691192627  21     0.05506539344787598  1.0638303756713867
15        22      1.4919278621673584  22     0.06217360496520996  1.4297542572021484
16        23      5.640883445739746   23     1.4453530311584473   4.195530414581299
17        24      9.390121698379517   24     0.10560035705566406  9.284521341323853
18        25      6.075580358505249   25     3.589163303375244    2.486417055130005
19        26      69.62260890007019   26     6.0296471118927      63.59296178817749
20        27      87.26527261734009   27     3.354966402053833    83.91030621528625
--------  ------  ------------------  -----  -------------------  ------------------
```

Figure 4: Results in instances 11-20 for the different models

2.5.1 Implicit domain limitations

Since all rectangles must fit in the limits, some of the literals must necessarily be true: given a rectangle ri and its width w_i and height h_i, for each w' in $(W-wi..W)$ and for each h' in $(H-hi..H)$: px_i_w' and py_i_h' hold.

2.5.2 Implicit positional relation constraint

A rectangle cannot be both left and not left wrt a different rectangle, and the same reasoning can be applied to the *under* relation. Clauses of the type $\neg r_i\_leftof\_rj \vee r_j\_leftof\_r_i$ and $\neg r_i\_under\_rj \vee r_j\_under\_r_i$ were added for each pair of rectangles.

2.5.3 Domain reduction of largest rectangle

Whilte it does not eliminate all possible symmetries, it is possible to limit the domain of the largest rectangle, so that all the mirrored possibilities are not considered.
Given the order in which the variables are declared, imagining that the search would proceed trying to first assign the lowest values, the largest rectangle domain was limited so that it would be placed in the upper-right quadrant.
Given the largest rectangle (by area) ri, and the limits $limit_x = \lceil (W - wi)/2 \rceil$ and $limit_y = \lceil (H - hi)/2 \rceil$, the following clause is added for each value e in $(0..limit_x)$ and f in $(0..limit_y)$: $\neg px_{ri}\_e$ and $\neg py_{ri}\_f$.
In addition, for each rectangle rj such that $wj \geq limit_x$ or $hj \geq limit_y$ the clauses $\neg r_i\_leftof\_r_j$ and $\neg r_i\_under\_r_j$ are added, respectively.

2.5.4 Positions of large dimension rectangles

Some rectangles have large dimensions, especially when considering their height. It is possible to reduce the domain of search in the case the sum of the dimensions of the rectangles would go out of border. For each of such couples, clauses of the type $\neg r_i\_leftof\_r_j$, $\neg r_j\_leftof\_r_i$ and $\neg r_i\_under\_r_j$, $\neg r_j\_under\_r_i$ were included.

2.5.5 Equal rectangles

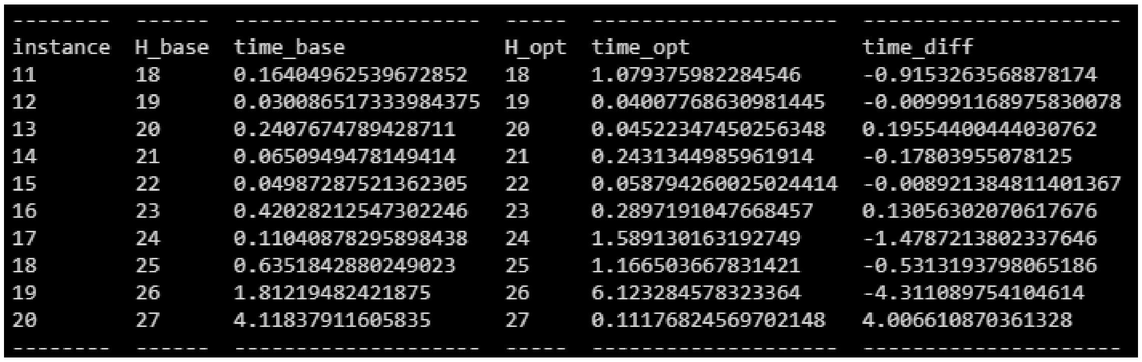
In the case rectangles of equal shapes are present, it is possible to reduce the search domain by fixing their relative positions, by adding the following clauses: $\neg r_j\_leftof\_r_i$ and $\neg r_j\_under\_ri \vee r_i\_leftof\_rj$.

2.6 Bisection algorithm for H_{min}

The optimal height is bound by the limits $ll = totarea/W$ (the containing rectangle must have area at least equal to the sum of the areas of the components) and $ul = 2 * max(h_{max}, totarea/W)$[3].
A bisection method is applied to search the possible H values, starting from ul, updating the ul and ll values depending on the satisfiability of the problem at searching at $H = (ll + ul)/2$, until $ul > ll$.
The main issue of this approach is that the model is recreated every time H is updated, losing the possibility of exploiting previously learned clauses, nonetheless, in most of the cases at hand the result was found to be reachable already at the first iterations.

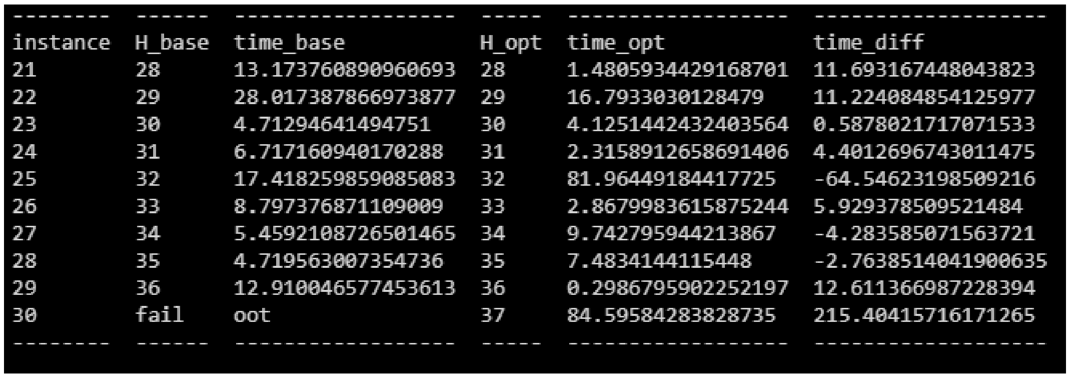
2.7 Results

Due to intrinsic non-deterministic decisions by z3, it was not possible to precisely evaluate the improvement in performance given by the optimization procedure, nonetheless the two models were compared on the instances 11-30: for the smaller instances, the optimized model does not seem to do particularly well, but for the more complex ones, it does show to add a general improvement in performance, with the exception of a couple of cases, likely due to the aforementioned non-deterministic decisions. In any case, since the loss in performance at smaller instances was not excessive, the optimized model was chosen and used to compute the final results.



```
--------  ------  --------------------  -----  --------------------  ---------------------
instance  H_base  time_base             H_opt  time_opt              time_diff
11        18      0.16404962539672852   18     1.079375982284546     -0.9153263568878174
12        19      0.030086517333984375  19     0.04007768630981445   -0.009991168975830078
13        20      0.2407674789428711    20     0.04522347450256348   0.19554400444030762
14        21      0.0650949478149414    21     0.2431344985961914    -0.17803955078125
15        22      0.04987287521362305   22     0.058794260025024414  -0.008921384811401367
16        23      0.42028212547302246   23     0.2897191047668457    0.13056302070617676
17        24      0.11040878295898438   24     1.589130163192749     -1.4787213802337646
18        25      0.6351842880249023    25     1.166503667831421     -0.5313193798065186
19        26      1.81219482421875      26     6.123284578323364     -4.311089754104614
20        27      4.11837911605835      27     0.11176824569702148   4.006610870361328
--------  ------  --------------------  -----  --------------------  ---------------------
```

Figure 5: Results in instances 11-20 for the 2 models



```
--------  ------  ------------------  -----  ------------------  -------------------
instance  H_base  time_base           H_opt  time_opt            time_diff
21        28      13.173760890960693  28     1.4805934429168701  11.693167448043823
22        29      28.017387866973877  29     16.7933030128479    11.224084854125977
23        30      4.71294641494751    30     4.1251442432403564  0.5878021717071533
24        31      6.717160940170288   31     2.3158912658691406  4.4012696743011475
25        32      17.418259859085083  32     81.96449184417725   -64.54623198509216
26        33      8.797376871109009   33     2.8679983615875244  5.929378509521484
27        34      5.4592108726501465  34     9.742795944213867   -4.283585071563721
28        35      4.719563007354736   35     7.4834144115448     -2.7638514041900635
29        36      12.910046577453613  36     0.2986795902252197  12.611366987228394
30        fail    oot                 37     84.59584283828735   215.40415716171265
--------  ------  ------------------  -----  ------------------  -------------------
```

Figure 6: Results in instances 21-30 for the 2 models

The results were computed running on colab environment. The solved instances can be found in the folder *SAT/output*, as it can be seen, the model managed to solve all but 3 of the instances(32,38,40). In all the other cases, the optimal H was found to be at the initial lower limit. It is likely that for the failed instances, the correct height would be found at further iterations of the process.

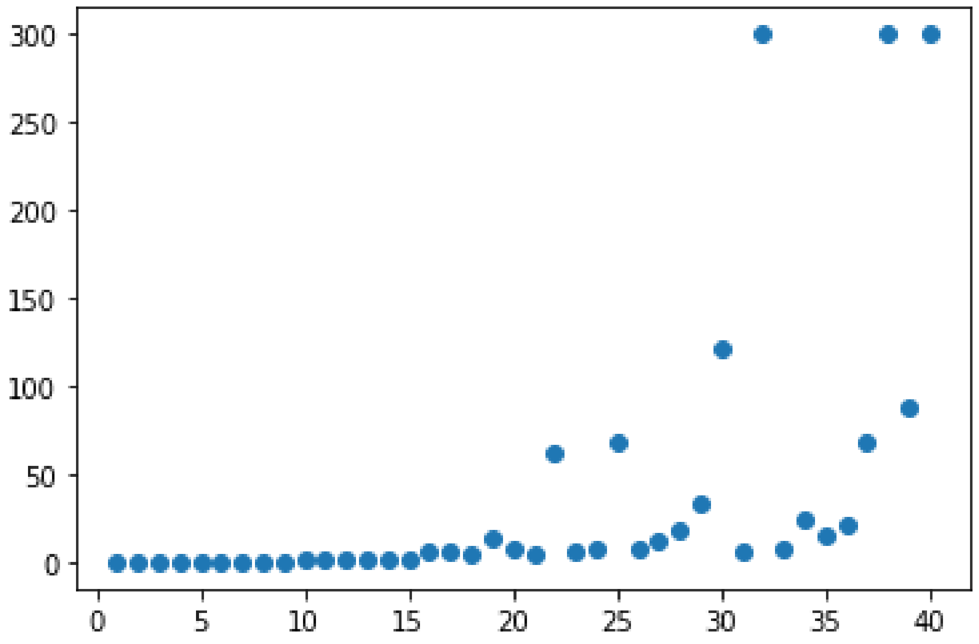


Figure 7: Time required for each instance

2.8 Suggestions for including rotation

If rotations are allowed the possible combinations of rectangles increase exponentially. In order to solve the problem, a possibility could be to consider an expanded list of rectangles, where all the rotated copies are included. The model requires a way to consider only one of each couple of rotated rectangles, so a list or *used* literal is implemented, with some new axioms and modifications to the previous model. The new axioms consist in the xor operation between pair of *used* variables, so that only one of each couple of rectangles will be placed. The rest of the code is kept virtually identical, with the exception of the non overlapping constraints: they will be valid only if the elements of the couple are both used.
A toy example is available in the notebook. It must be noted that the actual implementation of the suggestions above would require some other tweaks in order to work in any situation, since at the moment the model includes assumption on the variable bounds that are incompatible with the new approach.

3 SMT

In this section is discussed the model for VLSI in SMT and are shown the results obtained with it.

3.1 Model

The problem's formulation and constraints are similar to those of CP; in particular, given "sol" a 2D-array with $sol_{i,0}$ the x variable and $sol_{i,1}$ the y variable of the bottom-left corner of a circuit, we imposed that:

1. $sol_{i,0} \geq 0 \wedge sol_{i,0} \leq max\_width$, with $max\_width = \min(w, \sum_{c \in CIRCUITS} dims_{c,1})$;
2. $sol_{i,1} \geq 0 \wedge sol_{i,1} \leq max\_height$, with $max\_height = 2 * \max(\max(dim\_ver), min\_height)$, property shown by [3];
3. for the object function: $min\_height \leq obj \leq max\_height$ with $min\_height = \lfloor \frac{\sum_{c \in CIRCUITS} dims_{c,1} * dims_{c,2}}{w} \rfloor$, with w given width.

What changes are the encodings of the constraints for which exists a global version in Minizinc: diffn, symmetries and cumulatives.
diffn, which avoids intersections, has been written, as in the base model of Minizinc, $sol_{c1,0} + dims_{c1,0} \leq sol_{c2,0} \vee sol_{c2,0} + dims_{c2,0} \leq sol_{c1,0} \vee sol_{c1,1} + dims_{c1,1} \leq sol_{c2,1} \vee sol_{c2,1} + dims_{c2,1} \leq sol_{c1,1}$.

Two of the three symmetries have been written in the same way of cp:

1. For each pair of rectangles with only an equal dimension: $sol_{c1,x} == sol_{c2,x} \Rightarrow sol_{c1,y} \leq sol_{c2,y}$
2. Biggest rectangle fixed in the left bottom quadrant of the enclosing rectangle, i.e. constraint $sol_{0,0} \leq 1 + \frac{w - dims_{0,0}}{2} \wedge sol_{0,1} \leq 1 + \frac{max\_height - dims_{0,1}}{2}$

In the case of rectangles with both dimensions equal we have implemented a lexicographic symmetry breaking constraint through alpha M encoding [1], but the implemented version would have forced us to use an "and" between horizontal and vertical coordinates, so we opted for a model that uses constraints between pairs of circuits, i.e. $sol_{i,0} \leq sol_{j,0} \vee sol_{i,1} \leq sol_{j,1}$ for all circuits i and j with equal shape.

In order to try to get a more effective propagation we have implemented a decomposition of the cumulative constraint:

$$cap \geq \sum_{i \in CIRCUITS} (s_i \leq t \wedge t < s_i + d_i) * r_i \tag{11}$$

3.2 Results

In figure 8 is shown the plot of time/instance for the SMT model, as can be seen it requires more time then CP, this could be caused by the global constraints of Minizinc.
The results shown are taken with a model that uses the cumulative constraint to improve propagation, but we obtained better performances with a model without this two constraints.
A few instances couldn't be solved by the model.

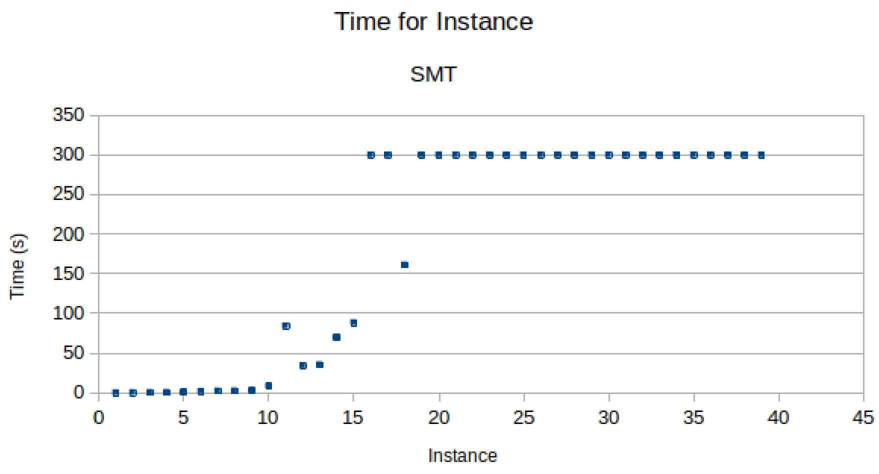


Figure 8: Model SMT, time required to solve each instance.

References

[1] Hani Abdalla Muftah Elgabou. *Encoding The Lexicographic Ordering Constraint in Satisfiability Modulo Theories.* URL: `https://www.cs.york.ac.uk/aig/constraints/SMT/ElgabouThesis.pdf`.

[2] Takehide Soh et al. "A SAT-based Method for Solving the Two-dimensional Strip Packing Problem". In: *Fundam. Inform.* 102 (Jan. 2010), pp. 467–487. DOI: `10.3233/FI-2010-314`.

[3] A. Stainberg. "A Strip-Packing Algorithm with Absolute Performance Bound 2". In: *SIAM Journal on Computing* 26 (Mar. 1997), pp. 401–409. DOI: `10.1137/S0097539793255801`.