Final Project of Combinatorial Decision Making and Optimization, Module 1

SMT

Federico Cichetti - federico.cichetti@studio.unibo.it

March 8, 2022

Contents

Introduction

This report describes the choices behind the implementation of different solutions for the Very Large Scale Integration (VLSI) problem, presented as a project work for the course of Combinatorial Decision Making and Optimization, Module 1.

This document in particular focuses on how the problem can be modeled into the *Satisfiability Modulo Theory* framework and solved efficiently by a SAT/SMT solver. Some of the concepts used in this solution have already been explained in both the CP and the SAT reports and are therefore not repeated here. Furthermore, our SMT solution can be seen as an almost direct translation of the CP solution, so for many similar concepts, references will be made to their explanation in the CP report.

1 Model

The SMT encoding of our problem has the advantage of being relatively simple to understand and express, but SMT solvers are not as efficient as SAT or CP solvers, so our solution is the least efficient between the three implemented for the project. Nevertheless, the amount of solved instances is comparable to the previously mentioned models.

1.1 Variables

Similarly to the CP model, for each instance to be solved the constants that are input for the model are integer numbers:

- W: The width of the plate, which is fixed
- n: The number of circuits to be placed
- The measures of the circuits, that we save into two arrays: cw for widths and ch for heights. The set of circuits is $C = [c_1, \ldots, c_n]$.
- The initial solution, obtained with the algorithm we have described in the CP report. Like for SAT, the SMT solver has no warm start mechanism, so we use the initial solution just to have an *higher bound* on the height of the circuit: $maxh$.
- Also, like we have done for CP and SAT we can provide a lower bound to the height by summing all circuits' areas and dividing by W. This lower bound is referred to as $lowh$ and is formally defined as $lowh = \lfloor \frac{\sum_i ch_i cw_i}{W} \rfloor$.

Similarly to CP, in our model variables can also be real or integer numbers, which is a great advantage over SAT, whose requirement of only using boolean variables made the model quite complex to utilize, explain and extend. The variables we are interested in finding values for are:

- h: The height of the plate. It's an integer number whose domain is $lowh <= h <= maxh$.
- cx: The 1D array of x-coordinates of the bottom-left corner for each of the $|C|$ circuits. It's an integer variable with domain: $\forall c \in C, 0 <= cx_c <= w$
- cy: The 1D array of y-coordinates of the bottom-left corner for each of the $|C|$ circuits. It's an integer variable with domain: $\forall c \in C, 0 <= cy_c <= maxh$

1.2 Constraints

As we highlighted in all previous reports, the main constraints of the problem regard two main aspects of any solution:

- The circuits should be placed *entirely* within the W and h boundaries of the board.
- The circuits should not overlap.

1.2.1 Respecting Boundaries

We deal with the first requirement by restricting the domains of our variables:

$$\bigwedge_{c \in C} (cx_c \geq 0) \wedge (cx_c \leq W - cw_c) \tag{1}$$

$$\bigwedge_{c \in C} (cy_c \geq 0) \wedge (cy_c \leq h - ch_c) \tag{2}$$

We enforced these constraints by adding a custom implementation of two *cumulative constraints*. In the cumulative global constraint we talk about *scheduling tasks*, each having a *start time* (s) and a certain *duration* (d). There is an upper bound t_{max} which represents the latest time a task can end. For the entire course of their execution, tasks occupy some resource quantity, expressed by their *resource requirements* (r) and our goal is to schedule tasks so that these requirements never exceed a fixed higher bound b at any time.

The constraint asks that at any moment t within bounds, the sum of the resource requirements of the constraints that are being executed in that time is less or equal than the global bound b.

$$\forall t \in [0, \ldots, t_{max-1}] : \sum_{i \in [1,\ldots,n]: s[i] \leq t < s[i]+d[i]} r[i] \leq b \tag{3}$$

We implemented the cumulative constraint as a function *cumulativez3* and added the following constraints:

$$cumulativez3(cy, ch, cw, W) \tag{4}$$

$$cumulativez3(cx, cw, ch, h) \tag{5}$$

We use the positions on the two axis as start times, the heights and the widths as durations and the other dimension as resource requirement. The bound b is the maximal limit on the axis of the resource requirement. In the first case, the maximal time t_{max} at which a task can end (i.e. the maximal x or y coordinate at which a circuit can be placed) is $maxh$, in the second case it's W.

1.2.2 Bidimensional No-Overlap

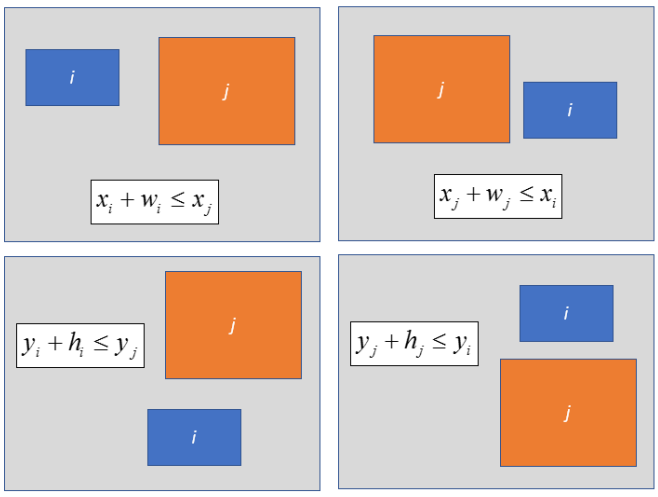

Figure 1: Decomposition of the no-overlap constraint: at least one of these situations must be verified for all pairs of circuits. **Image source**.

The second requirement is also known as *bidimensional no-overlap constraint* and is commonly decomposed in this way:

$$\bigwedge_{c1,c2 \in C, c1<c2} (xc_{c1} + cw_{c1} <= xc_{c2}) \vee (xc_{c2} + cw_{c2} <= xc_{c1}) \vee (yc_{c1} + ch_{c1} <= cy_{c2}) \vee (cy_{c2} + ch_{c2} <= cy_{c1}) \quad (6)$$

The idea is simply that at least one of the situations described in the constraint must be verified: the second circuit should start before or after the first one in both axes. Figure 1 shows a graphical representation of the decomposition. Unlike for our CP model where we used a global constraint instead, we decided to directly use this decomposition for simplicity.

1.2.3 Dual View

As for CP, we introduced a dual view into the problem with the array of integer variables tr, representing the starting position of circuits if the board was a flattened 1-dimensional array. We linked these variables to the positions cx and cy through the following channeling constraint:

$$\bigwedge_{c \in C} (tr_c = cy_c \times W + cx_c) \quad (7)$$

We used this dual view to pose some implied constraints: all starting positions should be different and *exactly one* of the circuits should be placed at (0,0):

$$Distinct_{c \in C}(tr_c) \quad (8)$$

$$exactly\_one_{c \in C}(tr_c = 0) \quad (9)$$

The first constraint is a built-in in our constraint solver, but it trivially expands to a set of inequalities. The second constraint is the same that we used in our SAT model and is implemented as a combination of *at most one* and *at least one* constraints.

1.2.4 Symmetry Breaking Constraints

The dual view has the benefit to express circuits position in a 1D array, which is a structure for which defining a lexicographic ordering is trivial. Therefore, we decided to once again use the symmetry breaking strategy explained in the CP report that uses *lexicographic comparisons* to eliminate the three main kinds of symmetry of the model: vertical, horizontal and diagonal flippings.

We implemented the $lex \leq$ constraint in SMT by comparing the single elements of two arrays of variables:

$$lex \leq (vars1, vars2) \leftrightarrow (vars1_1 \leq vars2_1) \wedge \left(\bigwedge_{i \in [1,\ldots,|vars1|-1]} (vars1_i = vars2_i \rightarrow vars1_{i+1} \leq vars2_{i+1}) \right) \quad (10)$$

As we already discussed, in a horizontal flipping, circuit c in position (cx_c, cy_c) moves at position $(W - cx_c - cw_c, cy_c)$, while in a vertical flipping, the same block moves at position $(cx_c, h - cy_c - ch_c)$. Flipping in both axes maps circuit c to position $(W - cx_c - cw_c, h - cy_c - ch_c)$.

Therefore, we added the following constraints to the problem:

$$lex \leq (tr, [cy_c \times W + (W - cx_c - cw_c) | c \in C]) \quad (11)$$

$$lex \leq (tr, [(h - cy_c - ch_c) \times W + cx_c | c \in C]) \quad (12)$$

$$lex \leq (tr, [(h - cy_c - ch_c) \times W + (W - cx_c - cw_c) | c \in C]) \quad (13)$$

2 Search

The objective of the search is to *minimize* h, the total height of the board. We have restricted the possible domain of h expressing its bounds:

$$(h \geq lowh) \wedge (h \leq maxh) \tag{14}$$

Furthermore, h is defined as the maximum between heights that have been reached by the circuits within the board.

$$h = \max_{c \in C}(cy_c + ch_c) \tag{15}$$

In order to find the lowest possible height, we framed the search as an *optimization problem*, where the objective to *minimize* is variable h (and in turn the y-coordinate of circuits positions). Our SMT solver efficiently implements a mechanism that allows optimization, so we directly used it rather than defining our own optimization algorithm like we did for SAT.

Unfortunately, the search for optimal solutions in SMT is the slowest among the models we implemented for the project. Since we are not able to customize the search with additional annotations, nor provide an initial solution as a warm start mechanism, the only way we can act for building a more efficient model would be by adding stronger constraints, which we expect to be our main focus in possible future works.

3 Rotation Extension

We did not implement a model that also allows rotation for SMT, but the general ideas we highlighted for the CP model should apply with almost no difference.

Rather than considering widths and heights of circuits as constants, we should model these entities as *variables*, so that the model can independently decide which measure is the current height and which is the width of the circuit for the solution. Therefore, we should introduce two arrays of variables, *curcw* and *curch* representing the current width and height of circuits. We can then post as constraints the fact that the solver should pick values for the current measures from the constants of that circuit, and that picking one of the measures for a circuit implies that the other one is assigned to the other dimension:

$$\bigwedge_{c \in C}((curch_c = ch_c) \vee (curch_c = cw_c)) \tag{16}$$

$$\bigwedge_{c \in C}(curch_c = ch_c \leftrightarrow curcw_c = cw_c) \wedge (curch_c = cw_c \leftrightarrow curcw_c = ch_c) \tag{17}$$

Then, we can simply change all instances of ch_c and cw_c in the other constraints with $curch_c$ and $curcw_c$ and the solver should be able to come up with sensible choices for rotations of circuits on its own.

4 Implementation

The model has been implemented in Python using the Z3 Python API for expressing the variables and constraints. As usual, we have built a *launcher* which contains the code for loading the instances specified by the user, for instancing the solver, the variables and constraints and for saving the solution as well as optionally displaying it graphically. The algorithms that provide the initial solution and sort the circuits by their size have been re-used from the CP program implementation.

The *cumulativez*3, $lex \geq$ and max functions had to be implemented using the Z3 API. Z3 has no way to determine to lower/upper bound of a decision variable at run-time, so to pose the *cumulativez*3 constraint (where we would need the lower and upper bounds for the positions that can be occupied by the circuits)

we pass to the function as additional arguments the lower and upper bounds to be used in each case (0 and $maxh$ for constraint 4 and 0 and W for constraint 5).

In SAT we used a `Solver()` object (from the Z3 API) for building the solution because we only needed to check the problem's satisfiability. Instead, for SMT we use the `Optimize()` object, to which we have specified h as the objective to *minimize*. The solver takes care of the optimization part completely on its own and returns an optimal solution together with the solving status. We set an internal timer to stop search after 5 minutes with the `timeout` configuration option.

The output of the solving process (when the constraints have been found to be satisfiable) is a mapping from each of the instanced variables to their chosen values for the solution. Differently from the SAT model, these values can also be integers, but we still need to apply some post-processing to obtain the correct value for each of the variables in the model.

As for the SAT models, in the solving report of Table 1, we add the actual solving time and the post-processing times together, because the post-processing step is fundamental for the coherence of the solution.

5 Results

We managed to solve 33 of the provided instances within the 5 minutes limit using our model. In terms of solved instances, the SMT model is slightly better than the CP model and slightly worse than SAT, but all solved instances generally took more time to solve with respect to our other models, so we believe that all things considered this is the model that needs more tuning in future works. One could argue that the simplicity of modeling with SMT can make up for the poor efficiency, but we feel like stronger constraints can greatly help the solving process. Table 1 shows the results and solution times we have obtained for each of the 40 instances.

Instance	Solving Time (s)	Solved
1	0.044004	True
2	0.046875	True
3	0.062502	True
4	0.140621	True
5	0.25	True
6	0.406249	True
7	0.468749	True
8	0.953123	True
9	0.828126	True
10	1.698	True
11	7.460999	True
12	5.454393	True
13	4.366367	True
14	5.507165	True
15	8.36564	True
16	22.330656	True
17	21.239123	True
18	32.023786	True
19	60.33109	True
20	49.374544	True
21	133.562085	True
22	187.441763	True
23	56.063638	True
24	40.198088	True
25	>300	False
26	185.792379	True
27	111.876736	True
28	88.025331	True
29	162.757111	True
30	>300	False
31	48.358662	True
32	>300	False
33	97.405405	True
34	68.242159	True
35	85.823317	True
36	101.924827	True
37	>300	False
38	>300	False
39	>300	False
40	>300	False

Table 1: Solving times and optimality status achieved by our model on the 40 instances of the problem. Times don't include the overhead of the Python launcher nor the time for finding an initial solution, but in most cases they are negligible.