VLSI Design: a CP approach

Davide Baldelli, davide.baldelli4@studio.unibo.it
Antonio Morelli, antonio.morelli3@studio.unibo.it
Tommaso Cortecchia, tommaso.cortecchia@studio.unibo.it
Stefano Ciapponi, stefano.ciapponi@studio.unibo.it

***Abstract*— This report describes a Combinatorial Optimization approach to the Very Large Scale Integration (VLSI) problem, exploiting Constraint Programming (CP) technologies.**

***Keywords*— VLSI, CP**

I. Introduction

VLSI (Very Large Scale Integration) 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 formal problem is designed as follows: 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). Consider two variants of the problem. In the first, each circuit must be placed in a fixed orientation with respect to the others. This means that, an $n \times m$ circuit cannot be positioned as an $m \times n$ circuit in the silicon plate. In the second case, the rotation is allowed, which means that an $n \times m$ circuit can be positioned either as it is or as $m \times n$.

An instance of VLSI is the width of the silicon plate w, the number of circuits n, and the horizontal and vertical dimension w_i and h_i of the i-th circuit. The solution should indicate the length of the plate l, as well as the position of each circuit by its x_i and y_i, which are the coordinates of the left-bottom corner.

The purpose of this project is to model and solve the problem using Constraint Programming (CP), propositional SATisfiability (SAT), its extension to Satisfiability Modulo Theories (SMT), and Linear Programming (LP). Solving processes that exceed a time limit of 5 minutes (300 secs) have been aborted.

II. CP

To model the CP solution we have used the solver Chuffed as a backend to the Minizinc constraint modelling language. Chuffed is a state of the art lazy clause solver designed from the ground up with lazy clause generation in mind.

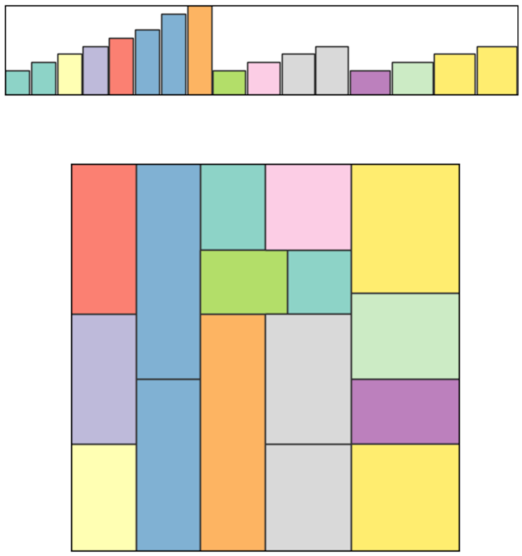

Fig. 1. Graphical representation of an instance and its solution.

A. Parameters

We receive an input in the form:

- w: width of the silicon plate;
- n: number of circuits to be placed;
- (w_i, h_i): width and height of the i-th circuit;

B. Variables

We have modelled the problem defining the following variables:

- h: height of the silicon plate, variable to minimize.
- (x_i, y_i) : position of the bottom-left corner of the circuits;

C. Constraints

The two main constraints force the circuits to fit inside the silicon plate:

$$x_i + w_i \leq w \qquad \forall\ 1 \leq i \leq n;$$

and impose no-overlap relationship between circuits, that is, a constraint imposing that each cell of the plate must be occupied by a single circuit. This constraint is expressed by imposing that for each couple of circuits, one must be above,

under, on the left, or on the right of the other. In Minizinc this can be accomplished by the global constraint `diffn`.

We have added some implied constraints that improved the performances of the model: we exploited the parallelism with the well-studied scheduling problem and impose the `cumulative` global constraint of Minizinc. Looking at the task as a resource usage problem each circuit has been considered as an activity whose duration is the height, its amount of resources is equals to its width while the total number of resources is w. The same reason can be done in the opposite sense, so each circuit has an activity whose duration is represented by the width and amount of resources is equals to its height and the total number of resources is h.

We have imposed the left-bottom corner of the largest circuit (the one with the maximum area) to lie in the bottom-left position with reference to the horizontal and vertical symmetries:

$$\hat{i} = argmax_{1 \leq i \leq n}(x_i \cdot y_i)$$
$$x_{\hat{i}} \leq (w - w_{\hat{i}})/2$$
$$y_{\hat{i}} \leq (h - h_{\hat{i}})/2$$

In this way we break the symmetries to the horizontal central axis and the vertical central axis (always but when the largest circuit is cut in half by an axis).

In order to make the model understand more rapidly when a solution is optimal we have added a lower bound to h, by imposing that the cells left empty must be at least zero:

$$w \cdot h - \sum_{1 \leq i \leq n} w_i \cdot h_i \geq 0$$

D. Search Strategy

In order to take full advantage of Chuffed's performance, it is often useful to add a search annotation to the model, but, as suggested in the Minizinc official documentation, to allow Chuffed to switch between this defined search and its activity-based search. In order to enable this behaviour, we used the Free search option in the solver configuration. We have used the Priority Search annotation [1] . An annotation of the form `priority_search(selvars, searches, varsel)` considers the array of variables `selvars` using the variable selection strategy `varsel` to select a variable, or equivalently an index into the array `selvars`. The index is then used to determine which of the array of search strategies `searches` is executed next. We have used as `selvars` the areas of the circuits, as `searches`, (y_i, x_i) with `indomain_min` as search strategy, and as `varsel`, `largest`. It means that the model selects first the circuits with the largest area, and select from its possible position the lowest, and between the lowest, the left-most. The intuition behind this is that it's more "difficult" to place bigger circuits as they occupy more space, thus we want our solvers to deal with them first, in particular by positioning them down in the silicon plate, and then move on the easier, smaller circuits.

E. Rotation model

To deal with the variation of the problem that allows the rotation of the circuits, we simply have defined a new boolean array f such that if f_i is True, then we swap the coordinates of the i-th circuit. We add a new simmetry breaking constraint that imposes the square circuits to not flip, as it would be meaningless, and we left everything else unvaried.

F. Results

The model revealed to be very effective and it solved all but the last of the proposed instances, and it finds a slightly suboptimal solution for the last instance as shown in Fig. 2. As we can see from the histogram in the Appendix, the performances do not change considerably between the two versions of the problem. Indeed, the solutions found are exactly the same for the two variants of the problem as, even not rotating the circuits, each of the first 39 solutions reaches the lower bound for h.

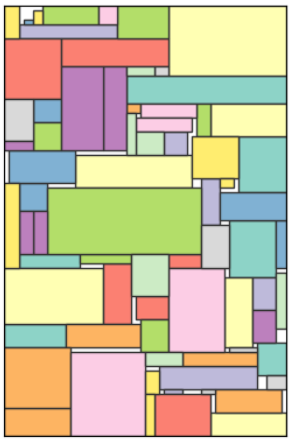

Fig. 2. Graphical representation of the suboptimal solution of the 40^{th} instance from the CP model.

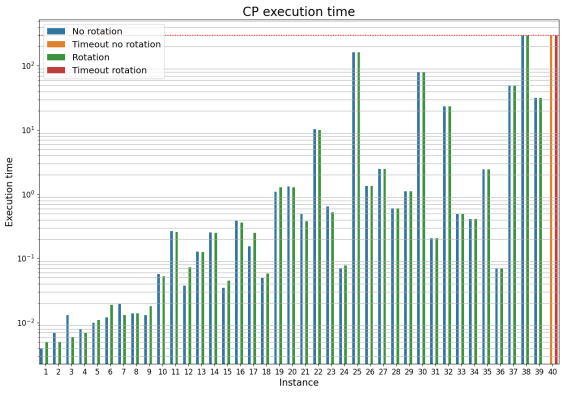

Fig. 3. Performance of the CP model (time in logit scale).

REFERENCES

[1] Feydy, Thibaut, et al. "Priority search with MiniZinc." ModRef 2017: The Sixteenth International Workshop on Constraint Modelling and Reformulation at CP2017. 2017.