VLSI SAT

Luca Fabri, luca.fabri@studio.unibo.it
Giuseppe Morgese, giuseppe.morgese2@studio.unibo.it

September 5, 2022

1 Introduction

The Boolean Satisfability Problem (SAT) is the problem of determining if a boolean expression is satisfiable. A boolean expression consists of a set of boolean variables, logical operators and parethesis and it is said satisfiable if it is possible to find an assignment to the variables which leads the formula to be True.

The logical operators are:

- $\neg$ NOT, $\vee$ OR (conjunction), $\wedge$ AND (disjunction)
- Implication: $p \implies q \equiv p \vee q$
- Bi-implication: $p \iff q \equiv (p \implies q) \wedge (q \implies p)$

In order to solve a SAT problem, modeling the variables and the constraints is required, and the problem must be expressed in Conjunctive Normal Form (CNF). Given a boolean variable X_1 (literal), a formula is in CNF if it's composed of a conjuction of disjunctions of literals e.g. a logical AND of logical ORs. For example, the following formula is expressed in CNF:

$$(X_1 \vee \neg X_3) \wedge (X_1 \vee X_2 \vee X_3) \wedge \neg X_2$$

Many state of the art SAT solvers are able to convert an expression into CNF formula and it has been proved that an arbitrary propositional formula can be converted to an equivalent expression into CNF: the Tseitin transformation is one of the available methods.

In many optimization problems, the search space the solver must explore is too big, therefore, one of the major challenges is to define constraints that could reduce it in order to allow the solver to perform less assignments. This leads to better implementations since the computational costs could be slightly reduced.

2 Modeling

In the literature, the VLSI problem could be modeled as the 2-dimensional Strip Packing Problem (2SPP). Given a set of rectangles (circuits) of dimension (w, h) and a strip of

bounded width and variable height (W, H), the optimization problem asks to place all the rectangles inside the strip such that they don't overlap and the height is minimized.

The SAT modeling we were inspired by has been taken from a paper of T.Soh et.al. (2010) [1] and the main concepts behind it will be explored here.

2.1 Defining lower and upper bounds

Since in SAT formulation it's impossible to have an objective function to minimize, it's necessary to define the plate's height lower and upper bounds.

The two metrics the closer they are the better, since the combinations of values the solver must explore are less. The lower bound metric has been taken from a paper of Vigo et.al. and it's defined as the maximum between the maximum circuit height and *continuous lower bound* [2]:

$$L_c = \left\lceil \frac{\sum_{i=0}^{n-1} w_i * h_i}{W} \right\rceil$$

$$LB = \max\{\max h, L_c\}$$

where n is the number of circuits.

On the other hand, the upper bound is defined as follows:

$$H = \left\lceil \frac{n * \max h}{\lfloor \frac{W}{\max w} \rfloor} \right\rceil$$

First, the minimum number of circuits placed one next to the other is computed (denominator), and it's assumed that this blocks are of height $\max h$. In the second step, the number of circuits is divided from the previous metric: this measure gives the number of blocks. In the end everything is multiplied by $\max h$ to obtain the upper bound.

2.2 Variable encoding

A project requirement is to return as output the left bottom coordinate of each circuit, to model them in the form of boolean variables, the px and py variables are introduced:

$$px_{i,c} \iff x_i \leq c$$

where $c \in \{0, \ldots, W-1\}$, $i \in \{0, \ldots, n-1\}$

$$py_{i,c} \iff y_i \leq c$$

where $c \in \{0, \ldots, H-1\}$, $i \in \{0, \ldots, n-1\}$.

Similarly, ph is used to encode the height. In particular, ph_o is True if all the circuits are packed downward o.

ph_i $i \in \{0, \ldots, H-LB-1\}$ is an array of length $H-LB$ and will help us in the execution time: when a specific height h is fixed, in order to check if the problem is SAT, the variable ph_{h-LB} is set to True.

The implied constraints of ph will be introduced in the next section.

In addition, the circuits mustn't overlap, so the following variables are also defined to specify if a circuits is placed before, next, at the top or at the bottom of another:

- $lr_{i,j}$ $\forall i, j \in \{0, \ldots, n-1\}, i \neq j$ is True if and only if the $i-$circuit is placed before the j's
- $ud_{i,j}$ $\forall i, j \in \{0, \ldots, n-1\}, i \neq j$ is True if and only if the $i-$circuit is placed at the top of the j's

2.3 Constraints

2.3.1 Ordering constraints

If $px_{i,c}$ is True, then $px_{i,c+1}$ is also True, and so on until c reaches $W-1$:

$$px_{i,c} \implies px_{i,e} \; \forall e \in \{c+1, \ldots, W-1\}$$

This equivalently holds for py and ph:

$$py_{i,c} \implies py_{i,f} \; \forall f \in \{c+1, \ldots, H-1\}$$

$$ph_o \implies ph_{o+i} \; \forall i \in \{o+1, \ldots, H-LB-1\}$$

2.3.2 Non overlapping constraints

The circuits can't overlap so it must hold:

$$lr_{i,j} \vee lr_{j,i} \vee ud_{i,j} \vee ud_{j,i} \quad \forall i, j \in \{0, \ldots, n-1\}, i < j$$

From this variables it's necessary to define a set of implied constraints, to limit the px and py variables once one of them is satisfied.

Following what's said in the ordering constraints, a circuit placed before the other ($lr_{i,j}$) implies that:

- If $px_{j,e+w_i}$ is True, surely $px_{i,e}$ is True:

$$lr_{i,j} \implies (px_{j,e+w_i} \implies px_{i,e}) \quad e \in \{0, \ldots, W-w_i-1\}$$

- At least for each $0 \leq e \leq w_i - 1$, $px_{j,e}$ is False:

$$lr_{i,j} \implies \neg px_{j,e} \quad e \in \{0, \ldots, w_i-1\}$$

similarly for the ud variables.

Putting everything into CNF, the constraints could be written as:

$$\neg lr_{i,j} \vee \neg px_{j,e+w_i} \vee px_{i,e} \quad e \in \{0, \ldots, W - w_i - 1\}$$

$$\neg lr_{i,j} \vee \neg px_{j,e} \quad e \in \{0, \ldots, w_i - 1\}$$

Briefly, all the non-overlapping constraints, lr and ud together, are the following. $\forall i, j \in \{0, \ldots, n-1\}, i < j$:

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

$$\neg lr_{i,j} \vee \neg px_{j,e+w_i} \vee px_{i,e} \quad e \in \{0, \ldots, W - w_i - 1\}$$

$$\neg lr_{j,i} \vee \neg px_{i,e+w_j} \vee px_{j,e} \quad e \in \{0, \ldots, W - w_j - 1\}$$

$$\neg ud_{i,j} \vee \neg py_{j,f+h_i} \vee py_{i,f} \quad f \in \{0, \ldots, H - h_i - 1\}$$

$$\neg ud_{j,i} \vee \neg py_{i,f+h_j} \vee py_{j,f} \quad f \in \{0, \ldots, H - h_j - 1\}$$

$$\neg lr_{i,j} \vee \neg px_{j,e} \quad e \in \{0, \ldots, w_i - 1\}$$

$$\neg lr_{j,i} \vee \neg px_{i,e} \quad e \in \{0, \ldots, w_j - 1\}$$

$$\neg ud_{i,j} \vee \neg py_{j,f} \quad f \in \{0, \ldots, h_i - 1\}$$

$$\neg ud_{j,i} \vee \neg py_{i,f} \quad f \in \{0, \ldots, h_j - 1\}$$

2.4 Height constraints

As previously said, ph_o is True if all the circuits are packed downward o. To force this constraint it's enough to reduce the domain of each py variable.

In particular:

$$ph_o \implies py_{i,k} \quad k \in \{o - h_i, \ldots, H - 1\}, i \in \{0, \ldots, n-1\}$$

Into CNF:

$$\neg ph_o \vee py_{i,k} \quad k \in \{o - h_i, \ldots, H - 1\}, i \in \{0, \ldots, n-1\}$$

2.5 Symmetry breaking constraints

Symmetry breaking constraints is an important step to achieve good results. In some cases, without them the performances of a solver could be very slow and solutions of bigger instances couldn't be computed. In this section are exposed the adopted ones.

2.5.1 Packing the biggest circuit by area

The circuit with the maximum area could be packed in the left bottom area of the plate, so its domain is reduced:

$$m = \{i : (w_i * h_i \geq w_k * h_k \ \forall k \in \{0, \ldots, n-1\}, i \neq k) \ \forall i \in \{0, \ldots, n-1\}\}$$

$$px_{m,e} \quad e \in \left\{ \left\lfloor \frac{W - w_m}{2} \right\rfloor, W - 1 \right\}$$

$$py_{m,f} \quad f \in \left\{ \left\lfloor \frac{H - h_m}{2} \right\rfloor, H - 1 \right\}$$

Since the biggest circuit is placed at the left bottom area, all the circuits whose width exceeds the half of the remaining width of the plate cannot be placed before it:

$$w_i > \left\lfloor \frac{W - w_m}{2} \right\rfloor \implies \neg lr_{i,m} \quad i \in \{0, \ldots, n-1\}$$

2.5.2 Packing large rectangles

If the sum of the heights or the widths of a pair or rectangles exceeds the bounds, then they cannot be placed one next to the other, or one at the top of the other. $\forall \quad i, j \in \{0, \ldots, n-1\}, i < j$:

$$w_i + w_j > W \implies \neg lr_{i,j} \wedge \neg lr_{j,i}$$

$$h_i + h_j > H \implies \neg ud_{i,j} \wedge \neg ud_{j,i}$$

2.5.3 Packing same rectangles

If two rectangles have the same dimension e.g. $(w_i, h_i) = (w_j, h_j)$ we can fix the positional relation of the rectangles:

$$\neg lr_{j,i} \wedge lr_{i,j} \vee \neg ud_{j,i}$$

3 Development

The model implementation has been developed using the python API Z3Py.

Z3 is a Theorem Prover, engineered in the Research in Software Engineering (RiSE) group at Microsoft Research, used in many application such as software/hardware verification, constraint solving and security.

The API allows the creation of symbolic boolean variables and formula and it's able to automatically translate any boolean formula into CNF.

The variables and the constraints previously presented in the previous sections are all implemented and rotation could also be enabled.

3.1 Bisection method

Since in SAT it's impossible to define an objective function, the height the solver must check is fixed before the computation and the optimal one is found through the bisection method.

The bisection method works as follows:

- First, an initial height h is set.
- Then, the upper or lower bound are updated: if the problem is UNSAT, the lower bound LB is set to h, otherwise, the upper bound is reduced and it will be set equal to h.
- This second step is computed until the lower and upper bound will coincide: the optimal height is the final h.

3.2 Architecture

The whole code is organized in a way that the domain model (`Plate`, `Circuit`) is separated from the implementation of the solver (CP, SAT, LP).

In fact, the latter extend the `VLSISolver` class which containts an abstract method `vlsi_solve()`, in order to leave the implementation of the VLSI problem to a specific solver: ideally, any solver could be written, the only requirements is that an object of type `Plate` is returned.

The notebooks in the `src` directories are almost the same, the only thing that changes is the just mentioned `VLSISolver`: `VLSI_CP`, `VLSI_SAT`, `VLSI_LP`.

The SAT version contains also another solver, `VLSI_SAT_rot` which enables the rotation of the circuits.

In addition, to the `VLSI_SAT` constructor could be set the following flags to experiment the performance differences:

- `sym_br_enabled=False` to not add the symmetry breaking constraints (by default is True)
- `dom_red_enabled=False` to not add the domain reducing constraints such as the biggest circuit placing.

4 Results

This implementation of the VLSI problem, without rotation enabled, could solve in 5 min all the instances except the last in a machine equipped with Intel Core i7-8550U CPU @ 1.80GHz. However, due to a lack of time, we couldn't plot the benchmark of the SAT version.

5 Conclusions

The SAT version has been quite challenging. Finding the right documentation and test the results so that the solver could find the optimal height even for the largest instances led us to try different modeling approaches.

Moreover, the unexpected exit of the third colleague just over a week before the deadline has led to an increase of the load: the benchmarks of the SAT version lacks of the performance differences between with/without symmetry breaking and with/without domain reducing constraints.

References

[1] T.Soh et.al. *A SAT-based Method for Solving the Two-dimensional Strip Packing Problem*. URL: https://www.researchgate.net/profile/Naoyuki-Tamura/publication/220445013_A_SAT-based_Method_for_Solving_the_Two-dimensional_Strip_Packing_Problem/links/0912f50d6fba9a52f1000000/A-SAT-based-Method-for-Solving-the-Two-dimensional-Strip-Packing-Problem.pdf.

[2] D.Vigo S.Martello M.Monaci. *An Exact Approach to the Strip-Packing Problem*. URL: https://pubsonline.informs.org/doi/epdf/10.1287/ijoc.15.3.310.16082.