Report for Project Work
Combinatorial Decision Making and Optimization
Module 1
(SAT Solution)

Yuwei Ke (yuwei.ke@studio.unibo.it)
Yiran Zeng (yiran.zeng@studio.unibo.it)

Contents

1 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. In this project, a fixed-width plate and a list of rectangular circuits are given, all the circuits should be put on the plate and the height need to be minimized.

2 Implementation

The code is implemented in python with SAT solver. The very first part of the modeling process is to decide how to model the problem and to define the variables to be used. In SAT, the modeling process is done using propositional logic formulas consisting of Boolean variables and logical operators, which in this case are expressed using the relational operators provided by the Z3Py library.

2.1 parameters

2.1.1 Input

The input variables are the following ones:

w - the width of the silicon plate

n - the number of circuits

[$width_1$, $width_2$, .., $width_n$] - the width of every circuit

[$height_1$, $height_2$, .., $height_n$] - the height of every circuit

2.1.2 Output

2.1.2.1 Normal model output

Once the program finished optimization process and find solution, it will give output
as following:

w h

n

$Width_1$ $height_1$ x_1 y_1

$Width_2$ $height_2$ x_2 y_2

.....

$Width_n$ $height_n$ x_n y_n

Among these parameters:

- h is the minimum height the program needs to optimize.

- x_i and y_i correspond to the left bottom point's x-coordinate and y-coordinate of the i circuit.
- w, n, $width_i$, $height_i$ are same as the input.

And it will generate a picture as Figure 1:

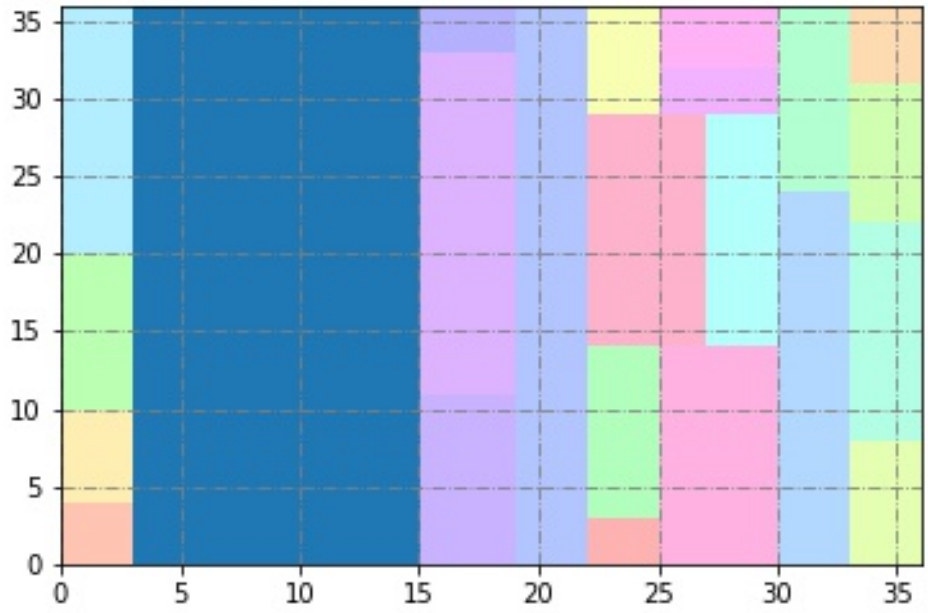

Figure 1: instance 29 with width=36, n=23 and height=36

2.1.2.2 Rotation model output

Output format is as following:

w h

n

$Width_1$ $height_1$ x_1 y_1 $Rotation_1$

$Width_2$ $height_2$ x_2 y_2 $Rotation_2$

.....

$Width_n$ $height_n$ x_n y_n $Rotation_2$

- Rotation is boolean variable.When it is true, it means the circuit has been rotated.On the contrary, false means it is not rotated.
- Other variables' meaning is same as 2.1.2.1
- It will also generate a picture like Figure 1.

3 Normal model Constraints

3.1 Implied constraint

When taking in to account this problem, we can not ignore the implied constraint of the problem itself. As the plate's width is fixed, all the circuits inside could not exceed the left, right and button boundary of the plate.In the solution, the left-bottom corner of plate is the coordinate's origin, and bottom edge is X axis and left edge is Y axis.So there should have formulae:

$$x_i + width_i \leq w(0 < i \leq n)$$
$$x_i \geq 0$$
$$y_i \geq 0$$
$$y_i + height_i \leq h$$

3.2 Global constraints

3.2.1 cumulative

In this strip packing problem, it can be treated as resource scheduling problem. Each circuit can be viewed as a task, the horizontal axis of the plate corresponds to the time, the vertical axis corresponds to the capacity, the start time and duration of each task can be viewed as the horizontal axis and width of the circuit, while the amount of resources required corresponds to the height. On the other hand, when the duration be represented by vertical axis and the amount of resource is showed by horizontal one, that is also reasonable. The function cumulative() has been realized :

```
def cumulative(start, duration,requirement,capacity):
    c = []
    for u in requirement:
        c.append(
            sum([If(And(start[i] <= u, u < start[i] + duration[i]), requirement[i], 0)
            for i in range(len(start))]) <= capacity
        )
    return c
```

So there have constrains like below:

$$cumulative(x, width, height, h)$$
$$cumulative(y, height, width, w)$$

3.2.2 No Overlapping

Circuits must not overlap with each other. That constraint basically states that circuit i needs to be either to the left, to the right, below or above circuit j. For each distinct pair of circuit i and j:

$$x_i + width_i \leq x_j$$
$$x_i \geq x_j + width_j$$
$$y_i + height_i \leq y_j$$
$$y_i \geq y_j + height_j$$

3.2.3 Remove gap

To make sure there is no gap between circuits which means that each circuit is complete fit on another piece of circuit so we also need to add some other constraint.

$$\left(x_i = 0 \bigvee x_i = x_j + width_j\right)$$
$$\bigwedge$$
$$\left((y_i = 0 \bigvee y_i = y_j + height_j\right)$$

3.3 Symmetry breaking constraints

Symmetry may lead to a solution/failure which will have many symmetrically equivalent states.For example, exchange circuits with same width and height; flip the plate horizontally or vertically; rotate the plate 180°. if these circuits swapped positions it would not change the outcome of the satisfiability check. This leads to the solver doing redundant tests which depending on the input circuit set may decrease the performance significantly.
In order to get higher efficiency, we need to reduce the solution and search space by defining some symmetry breaking constraints:

● Sort all circuits by area from largest to smallest, for more efficient, there only let the one with the largest area in the lower part of the second largest circuit. In this way, the circumstances including horizontal and vertical flip as well as 180° rotation are all ruled out. it has been implemented an adapted version of less_eq ordering constraint.

$$lex\_less(x, y) = (x_0 \leq y_0) \wedge$$
$$(x_0 = y_0 \Rightarrow x_1 \leq y_1) \wedge$$
$$(x_0 = y_0 \wedge x_1 = y_1 \Rightarrow x_2 \leq y_2) \wedge$$
$$\ldots$$
$$(x_0 = y_0 \wedge x_1 = y_1 \wedge \ldots \wedge x_{i-1} = y_{i-1} \Rightarrow x_i \leq y_i)$$

● To limit the switching of circuits with same width and height: If the width and height of circuit i and circuit j are the same it is implied that if their x's are equal then y_j has to be greater than y_i and if they aren't then x_j has to be greater than x_i. For every pair of circuits:

$$(width_i = width_j \wedge height_i = height_j)$$
$$\Rightarrow$$
$$If\ (x_j = x_i)\ then\ (y_j \geq y_i)\ else\ (x_j \geq x_i)$$

4 Rotation Model

In this model, the only difference with the normal model is that the circuit's width and height can be exchanged. So an array of boolean variable is added which named Rotation. If the rotation[i] is true then it means the i^{th} circuit has been rotated and its width has been swapped with height.
There will be a new implied limitation in this model: if a circuit's height is larger than the plate's width, then it can not be rotated. So in conclusion they can be written as below:

```
width_r = [If(And(rotation[i],width[i] != height[i],height[i] <= w),height[i],width[i])
for i in range(n)]
height_r = [If(And(rotation[i],width[i] != height[i],height[i] <= w),width[i],height[i])
for i in range(n)]
```

5 Search

By using the Z3 solver API, The command `Solver` creates a general purpose solver. Solvers maintain a set of formulas and supports satisfiability checking, and scope management: Formulas that are added under one scope can be retracted when the scope is popped.

In sat problem, we want to find the sat from a optimal solution which is understand as best fitting VLSI, we can simply try iterate all heights from “minHeight” to “maxHeight” to check whether there is a solution.

The “minHeight” is the sum of the square of all circuits divide to w (the total width of the plate) and the “maxHeight” is the sum of the height of all circuits.

We can first generate our solver and add all the constraints mentioned above. And then we use ‘push’ and ‘pop’ commands in a for-loop to add a new constraint indicating the height each loop so to explore different checks when different height defined.

6 Result

During the python program's operation, if an instance takes more than 5 minutes to find the best solution, the program will automatically stop and run the next instance.

From the chart below, we can find that normal model are more efficient than the rotation model, and the solution founded by the normal model is also more optimal than the rotation model.

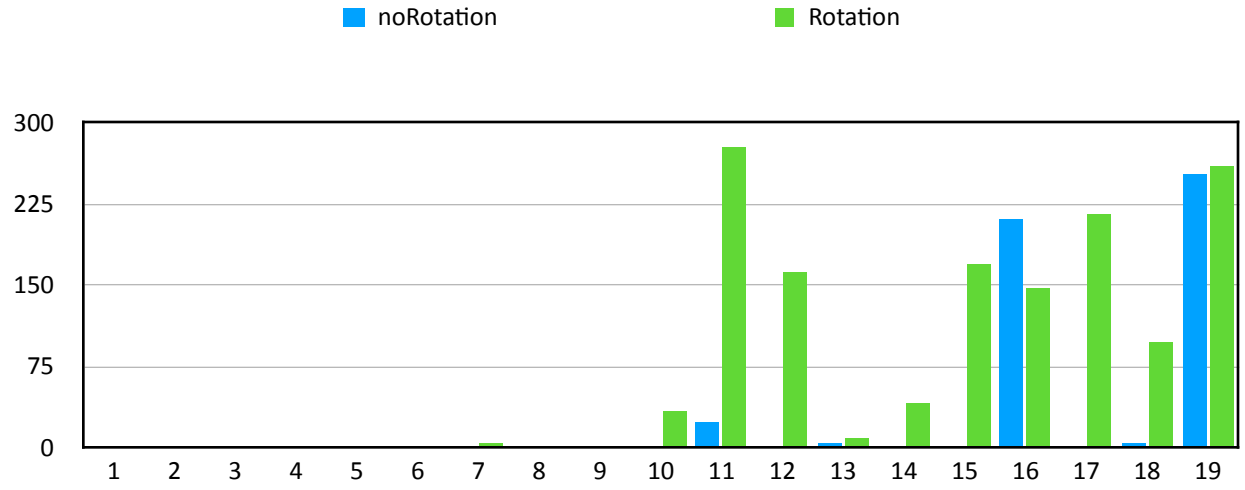

Figure 2: comparison between normal and rotation model on running time

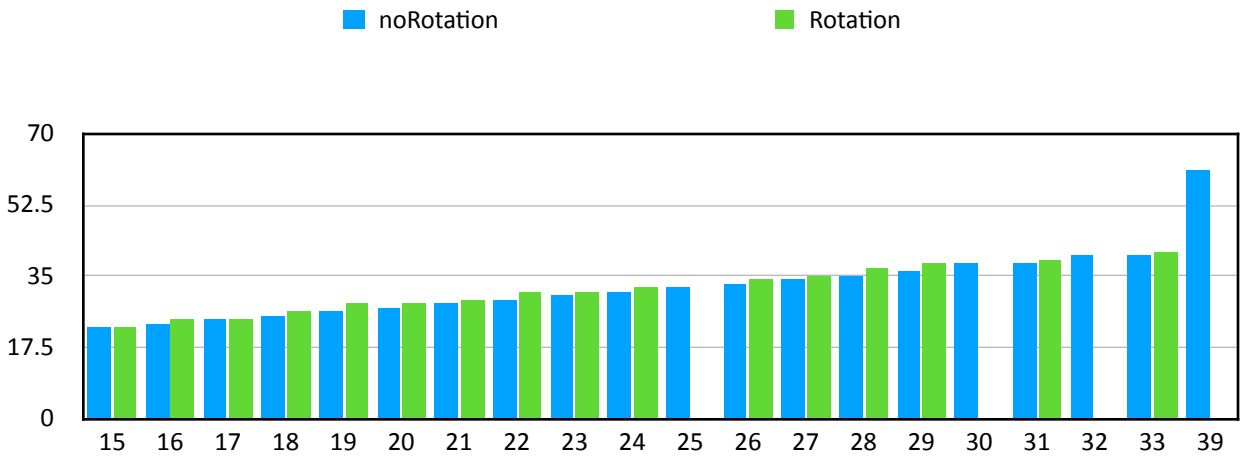

Figure 3: comparison between normal and rotation model on height size