SMT Report

Muhammad Zohaib Shafiq and Fabio Merizzi

Muhammad.shafiq6@studio.unibo.it

Fabio.merizzi@studio.unibo.it

August 2021

Contents

1 Introduction

In this report we will discuss the optimization of Very Larger Scale Integration (VLSI) optimization problem using python library Z3 based on Satisfiability Modulo Theory (SMT).

1.1 Python plotter

The solution set of the problem is visualised using same python program as for CP mention in Section 1.1 of CP-report. This surely helps us to identify errors from implementation and solution if any.

2 SMT

2.1 Parameters, Variables and Objective function

This section describes the relative parameters, variables and objective function required for the problem under consideration.

2.1.1 Parameters

The parameters defined in SMT are exactly same as in CP (MiniZinc) except CIRCUIT_DIM which is not defined/used here. Instead of explicitly defining this extra parameter we used python range function with num_circuits and data as parameter to define our range for respected x and y axis for variables and parameters where data is considered as whole input file.

```
#Input dimensions (x-axis,y-axis)
for i in range(2, len(data)):
    circuitX.append(int(data[i].split(' ')[0]))
    circuitY.append(int(data[i].split(' ')[1]))

#Variables (x-axis, y-axis)
X = [z3.Int('X_%s' % i) for i in range(num_circuits)]
Y = [z3.Int('Y_%s' % i) for i in range(num_circuits)]
```

Apart from this modification we have implemented python code to read input files from Instances (or any other) directory without having to make any single change in our input files keeping the same format.

2.1.2 Variables

Variables required for this problem are in total three (X, Y and Height) which are similar as described in Section 2.1 of CP-report, expect a change in the definition of their range.

```
X = [z3.Int('X_%s' % i) for i in range(num_circuits)]
Y = [z3.Int('Y_%s' % i) for i in range(num_circuits)]
Height = Int('Height')
```

X and Y variables are used to determine the starting or left most corner of the circuit placed in the silicon chip whereas Height illustrates the possible height for silicon chip after all the circuits have been placed.

2.1.3 Objective function

Objective is to find optimal minimum height by placing all the circuits given the fixed width of the silicon chip in advance.

```
s = Optimize()
s.add(constraints)
s.minimize(Height)
```

Since we cannot customise search strategy in SMT, we rely on the default search and it performs as good as expected.

2.2 Main problem constraints

In the following section we discuss the way we have encoded the constraints in SMT.

2.2.1 Upper and Lower Bound for Height

This constraint is very crucial for ending up with an optimized height. This sets the upper and lower bound for height, which is :

- upper bound, equal to the total sum of the dimensions of y-axis, in case if all the circuits are on the top of each other
- lower bound, equal to the height of the highest circuit in the current instance.

We may or may not define lower bound because it will never be lower than zero as we are to set many additional constraints in this regard. Not directly on height but on X and Y variables which will not let height be less than zero.

```
min_height = max(circuitY)
max_height = sum(circuitY)
ext = []
ext.append(And(Height>=min_height, Height<=max_height))
```

To express it in mathematical form we could use the following form:

Height $\geq min\_height \wedge Height \leq max\_height$

2.2.2 fixing the boundary of space

We must define this extra constraint in SMT because we found no way to implement it while declaring the variables as we did in CP (MiniZinc). We are defining range for variable X and Y in this domain. It does not allow variables X and Y to exceed height and width by keeping in view their starting point greater than or equal to zero.

```
domain = [ And(X[i] >= 0, X[i] < width, Y[i] >= 0, Y[i] < Height)
             for i in range(num_circuits)]
```

To express it in mathematical form we could use the following form:
n = num_circuits

$$\bigwedge_{i=0}^{n} (X[i] \geq 0 \wedge X[i] < width \wedge Y[i] \geq 0 \wedge Y[i] < Height)$$

2.2.3 ensures the fitting of circuits to chip dimension

This constraint allows the circuits to be fitted inside the silicon chip restricting it from going beyond the given range (width, height).

```
circuit_dim = [And(circuitX[i] + X[i] <= width, circuitY[i] +
Y[i]<=Height)for i in range(num_circuits) ]
```

To express it in mathematical form we could use the following :
n = num_circuits

$$\bigwedge_{i=0}^{n} (circuitX[i] + X[i] \leq width \wedge circuitY[i] + Y[i] \leq Height)$$

2.2.4 the non overlap constraint

Non overlap constraint ensures that no two circuits overlap. We first implemented this constraint using Distinct function of Z3 but it was unsuccessful.

```
non_overlap = []
for i in range(num_circuits):
  non_overlap.append(Distinct(X[i], Y[i], circuitX[i], circuitY[i]))
```

Therefore, we have implemented it in the same way as described in CP-report section 2.2.2.

```
non_overlap = []
for i in range(num_circuits):
    for j in range(num_circuits):
        if (i<j):
            non_overlap.append(Or(X[i] + circuitX[i]  <= X[j],
            X[j] + circuitX[j] <= X[i], Y[i] + circuitY[i] <= Y[j],
            Y[j] + circuitY[j] <= Y[i]))
```

To express it in mathematical form we could use the following form:
n = num_circuits

$\bigwedge_{i=0}^{n} \bigwedge_{j=0}^{n} if(i < j)(X[i] + circuitX[i] \leq X[j] \vee X[j] + circuitX[j] \leq X[i] \vee Y[i] + circuitY[i] \leq Y[j] \vee Y[j] + circuitY[j] \leq Y[i])$

2.2.5 the implied cumulative constraint

This constraint is again exactly implemented in the similar pattern as in our CP model to make sure that the sum of horizontal axis is at most to the width and similarly for vertical axis the sum is maximum up to the height.

```
implied = []
for i in range(width):
    for j in range(num_circuits):
        implied.append(Sum([If(And(X[j] <= i, i < X[j] + circuitX[j]),
        circuitY[j],0) for j in range(num_circuits)]) <= Height)

for i in range(num_circuits):
    for j in range(num_circuits):
```

```
implied.append(Sum([If(And(Y[j] <= i, i < Y[j] + circuitY[j]),
circuitX[j],0) for j in range(num_circuits)]) <= width)
```

To express it in mathematical form we could use the following form:
n = num_circuits

$\bigwedge_{i=0}^{n}\bigwedge_{j=0}^{n} if(X[j] \leq i \wedge i < X[j]+circuitX[j])then((\sum_{j=0}^{n} circuitY[j]) \leq Height)else0$
$\bigwedge_{i=0}^{n}\bigwedge_{j=0}^{n} if(Y[j] \leq i \wedge i < Y[j]+circuitY[j])then((\sum_{j=0}^{n} circuitX[j]) \leq width)else0$

2.2.6 symmetry breaking constraints

2.2.7 identical circuits

This constraint is useful in the cases where our model has to deal with circuits of the same dimension.

```
sym=[]
for i in range(num_circuits):
    for j in range(num_circuits):
        if (i<j):
            sym.append(Or(Not(And(circuitX[i] == circuitX[j],
            circuitY[i] == circuitY[j])), If(X[i] == X[j],
            Y[i] >= Y[j], X[i] > X[j])))
```

To express it in mathematical form we could use the following:
n = num_circuits

$\bigwedge_{i=0}^{n}\bigwedge_{j=0}^{n} if(i > j)then\neg(cicuitX[i] == circuitX[j] \wedge circuitY[i] == circuitY[j]) \vee$
$if(X[i] == X[j])then(Y[i] \geq Y[j])else(X[i] > X[j]))$

2.3 Rotation

The rotation part of our model can be easily implemented in SMT by making slight changes to our existing approach. The new approach has been clearly explained in CP-report section 2.5.

2.4 Conclusion

We consider the general performance of the SMT program to be satisfying, even if the number of instances solved by it is slighlty less than amount solved by the Minizinc program. Possibly this difference is due to the solver itself, which may be less efficient than Minizinc's solver, also, SMT has less possiblity of customization for the search strategies, and finally the constraints used in Miniznc were generally more optimized for the task.