Combinatorial Decision Making and Optimization

Project Report

VLSI Design

Gee Jun Hui Leonidas Yunani
Anna Fabris

Academic Year: 2020 / 2021

Contents

1 Introduction

The VLSI (Very Large Scale Integration) problem refers to the trend of integrating circuits into silicon chips. Such problems involve the development of search strategies to determine the optimal circuit placements on the silicon chips. Depending on the dimensions of the circuits and the silicon chip, the search space that must be traversed may be very large. As such, the program should be optimised to determine the circuit placements within a specified time constraint, which is the main objective of the project.

In this part, constraint programming (CP) via the MiniZinc language is used to model the problem. The CP model can be divided into two versions: CP (Normal) and CP (Rotation). CP (Normal) is a model where rotation of the circuits is not allowed, while CP (Rotation) is one where such rotations are permitted. Each model has been designed to use as many global constraints as possible. The model should then minimise the height of the silicon chip, while determining each circuit's placement on the chip.

2 Input & Output

The inputs to be fed to the program consists of a set of 40 text files. Each text file contains the following information:

- The max width of the silicon chip (`max_width`).
- The number of circuits (`n`).
- The width and height of each individual circuit ($W_i, H_i\ \forall i$ in $1, .., n$).

8 $\longleftarrow$ `max_width`
4 $\longleftarrow$ `n`
3 3 $\longleftarrow$ W_1 and H_1
3 5 $\longleftarrow$ W_2 and H_2
5 3 $\longleftarrow$ W_3 and H_3
5 5 $\longleftarrow$ W_4 and H_4

The program should read each text file and output the following information:

- The width and height of the silicon chip (`width`, `height`).
- The number of circuits (`n`).
- The width, height, lower left x-position and lower left y-position of each individual circuit ($W_i, H_i, Px_i, Py_i\ \forall i$ in $1, .., n$).

8 8 $\longleftarrow$ width and height
4 $\longleftarrow$ `n`
3 3 0 0 $\longleftarrow$ W_1, H_1, Px_1 and Py_1
3 5 0 3 $\longleftarrow$ W_2, H_2, Px_2 and Py_2
5 3 3 0 $\longleftarrow$ W_3, H_3, Px_3 and Py_3
5 5 3 3 $\longleftarrow$ W_4, H_4, Px_4 and Py_4

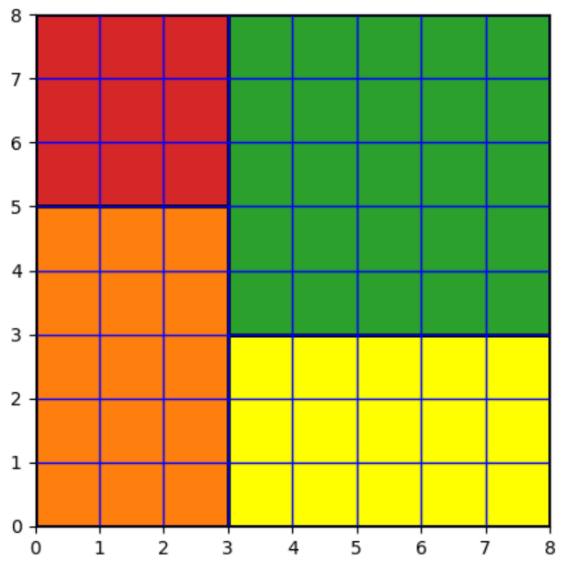

Figure 1: A visualization of a solved instance

Additionally, a visualisation of the solved instance is also saved as an image in the output folder.

3 Dependencies

The program has been written using the following software and libraries:

- The MiniZinc software for constraint programming.
- The Python MiniZinc library to allow for the interaction between Python and MiniZinc.
- The Matplotlib library to allow for the visualisation of the solved instances as part of the output.
- The tqdm library to allow for the tracking of the solving process for all instances.

The integration of MiniZinc with Python allows the program to efficiently read the input files, visualise each solved instance and output the final solutions with their respective images. As such, users can simply place the set of instances to be solved in the input folder and run the program from the command line interface.

4 CP (Normal)

4.1 Variables

To begin, the constants, decision variables and objective variable of the problem must first be defined. The **constants** can be thought of as parameters of the problem and have been defined as such:

- `circuit_heights` is an array of the *given* heights of all circuits.
- `circuit_widths` is an array of the *given* widths of all circuits.
- `max_height` is the *calculated* max height (upper bound) of the silicon chip. The max height is calculated by taking the sum of the given height for all the circuits.

  $$\sum_{i=1}^{n} H_i$$

  Where H_i is the given height of the circuit.
- `min_height` is the *calculated* min height (lower bound) of the silicon chip. The min height is calculated by taking the sum of the area of all the circuits and dividing it by the given max width of the silicon chip.

  $$A_i = W_i * H_i$$

  $$min\_height = \frac{\sum_{i=1}^{n} A_i}{max\_width}$$

  Where W_i is the given width of the circuit, H_i is the given height of the circuit and A_i is the calculated area of the circuit.
- `max_width` is the *given* max width of the silicon chip.

The **decision variables** represent the unknowns of the problem which must be determined by the model and are defined as follows:

- `start_x` is an array of x-coordinates of every circuit's lower left corner.
- `start_y` is an array of y-coordinates of every circuit's lower left corner.

The **objective variable** is the variable to be minimised by the model during search and is defined as follows:

- `height` is a variable whose value ranges from `min_height` to `max_height`.

4.2 Constraints

After defining the constants and variables of the problem, the constraints must be determined to shape the search space that will be traversed. The order in which the constraints are defined must also be selected carefully as they have a significant effect on the model's efficiency. As such the constraints have been defined and arranged in the following order:

- A `cumulative` constraint is used to determine the lower left y-coordinate of each circuit.

  ```
  constraint cumulative(
      start_y,
      circuit_heights,
      circuit_widths,
      max_width
  );
  ```

  Given the `circuit_heights` and `circuit_widths`, the `cumulative` constraint will stack the circuits vertically on top of each other such that they never exceed the given `max_width` of the chip.

  ```
  constraint forall(c in circuits)(
      start_y[c] + circuit_heights[c] <= height
  );
  ```

  Additionally, a `forall` constraint must also be defined to prevent the stacked circuits from exceeding the calculated `height` of the chip.

- A `cumulative` constraint is used to determine the lower left x-coordinate of each circuit.

  ```
  constraint cumulative(
      start_x,
      circuit_widths,
      circuit_heights,
      height
  );
  ```

  Given the `circuit_widths` and `circuit_heights`, the `cumulative` constraint will arrange the circuits horizontally side by side such that they never exceed the calculated `height` of the chip.

  ```
  constraint forall(c in circuits)(
      start_x[c] + circuit_widths[c] <= max_width
  );
  ```

  Additionally, a `forall` constraint is also defined to prevent the circuits from exceeding the given `max_width` of the chip.

- A `diffn` constraint is used to remove overlaps in the circuit placements on the chip.

  ```
  constraint diffn(
      start_x, start_y,
      circuit_widths, circuit_heights
  );
  ```

  `diffn` constrains the circuits by the coordinates (`start_x` and `start_y`) of their lower left corner and dimensions (`circuit_widths` and `circuit_heights`) to be non-overlapping.

4.3 Search Strategy

The search strategy choice will determine how the model propagates the constraints and assigns the values to each variable. The strategy has been defined as follows:

```
solve :: seq_search([
    int_search([height], dom_w_deg, indomain_min),
    int_search(start_y, dom_w_deg, indomain_min),
    int_search(start_x, dom_w_deg, indomain_min),
])
minimize height;
```

`dom_w_deg` will choose the variable with the smallest value of domain size divided by weighted degree, which is the number of times it has been in a constraint that caused failure earlier in the search.

`indomain_min` will assign the variable its smallest domain value. In the case of `height`, which must be minimised, `indomain_min` will cause the model to try values starting from the calculated `min_height`.

4.4 Results

The final model is able to solve 27/40 instances within a time constraint of 300 seconds. The model is able to easily solve the first 10 instances which are simpler by definition. It performs moderately well on instances between 11 and 30. However, it struggles to solve instances 31 and above which have many more circuits to consider.

5 CP (Rotation)

The CP (Rotation) model consists of two MiniZinc models: one for determining the required height, the other for packing the circuits on the chip based on the required height found. The 1^{st} model is relatively similar to the one used in CP (Normal) minus the packing of the circuits.

To ensure that the time constraint is still satisfied, the 1^{st} model is checked to see if it was completed within 5 minutes. If no, the instance is then considered a failure due to timeout. Otherwise, the remaining time is calculated and passed to the 2^{nd} model to pack the circuits.

5.1 Variables (Model 1)

Unlike the CP (Normal) model, the 1^{st} model for CP (Rotation) is only used for determining the required height of the chip. As such, the `start_x` decision variable, which represents the array of x-coordinates for each circuit's lower left corner, is not defined.

The listed variables below follow the same definition as their counterparts in the CP (Normal) model.

The **constants** are as follows:

- `circuit_heights`
- `circuit_widths`
- `max_height`
- `min_height`
- `max_width`

The **decision variable** is as follows:

- `start_y`

The **objective variable** is as follows:

- `height`

5.2 Constraints (Model 1)

The only constraint used for the 1^{st} model of CP (Rotation) is a `cumulative` constraint for determining the `height` of the chip. The `start_y` variable found also by the `cumulative` constraint is discarded.

Unlike the CP (Normal) model, an additional `cumulative` constraint for the `start_x` variable and a `diffn` constraint are not used.

```
constraint cumulative(
    start_y,
    circuit_heights,
    circuit_widths,
    max_width
);

constraint forall(c in circuits)(
    start_y[c] + circuit_heights[c] <= height
);
```

5.3 Search Strategy (Model 1)

The search strategy choice will only be applied to the `height` and `start_y` as `start_x` is not used for the 1st model of CP (Rotation).

```
solve :: seq_search([
    int_search([height], dom_w_deg, indomain_min),
    int_search(start_y, dom_w_deg, indomain_min),
])
minimize height;
```

5.4 Variables (Model 2)

The **constants** are as follows:

- `k` is the number of dimensions (in this case 2).
- `rect_size` is the size of each circuit in k dimensions.
- `rect_offset` is the offset of each circuit from the base position in k dimensions.
- `rect_shapes` is the set of rectangles defining the circuit's shape.
- `l` is an array of lower bounds defining the horizontal and vertical dimensions of the chip.
- `u` is an array of upper bounds defining the horizontal and vertical dimensions of the chip.
- `valid_shapes` is a valid shape that is assigned to each object among those available.

The **decision variables** are as follows:

- `x` is the x and y coordinates of each circuit's lower left corner.
- `kind` is the shape used by each circuit.

The values defined in `rect_size`, `rect_offset`, `rect_shapes` and `valid_shapes` will determine the final possible rotations of each circuit in `kind`.

An example on how to define the circuit rotations for an instance is given below:

```
8 ⟵ max_width
4 ⟵ n
3 3 ⟵ W1 and H1
3 5 ⟵ W2 and H2
5 3 ⟵ W3 and H3
5 5 ⟵ W4 and H4
```

The `rect_size` will be defined as follows:

```
rect_size = [|
    3, 3| % Squared-shaped circuit

    3, 5| % Rectangular-shaped circuit
    5, 3|

    5, 3| % Rectangular-shaped circuit
    3, 5|

    5, 5  % Squared-shaped circuit
|]
```

Here, each pair of values (e.g. 3, 5 and 5, 3) defines the shape that a circuit may take when rotation is allowed. Circuits which are square in shape will only have one pair of values assigned to them (e.g. 3, 3). This is because regardless of rotation, a squared-shaped circuit will occupy the same area.

The `rect_offset` is defined as follows:

```
rect_offset = [|
    0, 0| % Offset for 3, 3

    0, 0| % Offset for 3, 5
    0, 0| % Offset for 5, 3

    0, 0| % Offset for 5, 3
    0, 0| % Offset for 3, 5

    0, 0  % Offset for 5, 5
|]
```

Here, each offset is set to 0, 0 as the circuit shape starts from its lower left corner. Offsets can be used to define shapes such as tetrominoes (tetris blocks) where one rectangular part of the block might not start from the lower left corner of the tetrominoes.

The rect_shapes are then defined as follows:

```
rect_shapes = [
    {1}, % Possible shape of 3, 3

    {2}, % Possible shape of 3, 5
    {3}, % Possible shape of 5, 3

    {4}, % Possible shape of 5, 3
    {5}, % Possible shape of 3, 5

    {6}  % Possible shape of 5, 5
]
```

Here, each pair of values in rect_size is corresponds to a possible shape.

Finally, the valid_shapes are defined as follows:

```
valid_shapes = [
    {1},    % Possible shape for Circuit 1

    {2, 3}, % Possible shapes for Circuit 2

    {4, 5}, % Possible shapes for Circuit 3

    {6}     % Possible shape for Circuit 4
]
```

Here, valid_shapes determines the possible shapes from rect_shapes that each circuit may take when rotation is allowed.

5.5 Constraints (Model 2)

The constraints have been defined and arranged in the following order:

- A forall constraint is used to ensure that only circuits with certain rotations are accepted as valid as defined in the valid_shapes constant.

  ```
  constraint forall(obj in objects)(
      kind[obj] in valid_shapes[obj]
  );
  ```

- A geost_smallest_bb constraint is used to pack the circuits with rotations allowed into the chip.

```
constraint geost_smallest_bb(
    k,
    rect_size ,
    rect_offset ,
    rect_shapes ,
    x,
    kind ,
    l,
    u
);
```

The `geost_smallest_bb` constraint enforces that no two circuits overlap, and that all circuits fit within a global k dimensional bounding box. In addition, it enforces that the bounding box is the smallest one containing all circuits, i.e., each of the boundaries is touched by at least one circuit.

5.6 Search Strategy (Model 2)

The search strategy choice is only applied to the `kind` decision variable and has been defined as follows:

```
solve :: int_search(
    kind, dom_w_deg, indomain_min
)
satisfy;
```

A similar choice using `dom_w_deg` and `indomain_min` was also tested on the `x` decision variable. However, it was shown to only worsen the 2^{nd} model's performance.

5.7 Results

The final combined model is able to solve 15/40 instances with a time constraint of 300 seconds. The combined model is able to easily solve the first 10 instances which are simple by definition. However, it begins to struggle in solving instances 11 and above.