- **Algorithm Flow**

In this programming assignment, I apply the method "**Abacus**" [1] to legalize all the standard cells. Below is the approach "Abacus" [1].

```
1  Sort cells according to x-position;
2  foreach cell i do
3  |  c_best ← ∞;
4  |  foreach row r do
5  |  |  Insert cell i into row r;
6  |  |  PlaceRow r (trial);
7  |  |  Determine cost c;
8  |  |  if c < c_best then c_best = c, r_best = r;
9  |  |  Remove cell i from row r;
10 |  end
11 |  Insert Cell i to row r_best;
12 |  PlaceRow r_best (final);
13 end
```

First, sort all the cells according to its left x-coordinate (line 1). Then, the dynamic programming (line 2-13) begins to decide the best row for each cell to be inserted to according to the previous result. In order to determine which row is the best for the cell i, we insert the cell i into every row and determine the cost. Eventually, insert the cell i to best row (with least cost). Moreover, once the cell is inserted to the best row, it will never be moved to other rows. By the way, the cost is the total displacement of all already placed cells.

The most important thing is that how we can determine the x-position of all the cells in row r after inserting cell i into row r. The function "**PlaceRow**" deals with this problem. Below shows the function PlaceRow.[1]

```
   // Determine clusters and their optimal positions x_c(c):
1  for i = 1, ..., N_r do
2  |  c ← Last cluster;
   |  // First cell or cell i does not overlap with last
   |     cluster:
3  |  if i = 1 or x_c(c) + w_c(c) ≤ x'(i) then
4  |  |  Create new cluster c;
5  |  |  Init e_c(c), w_c(c), q_c(c) to zero;
6  |  |  x_c(c) ← x'(i);
7  |  |  n_first(c) ← i;
8  |  |  AddCell(c, i);
9  |  else
10 |  |  AddCell(c, i);
11 |  |  Collapse(c);
12 |  end
13 end
   // Transform cluster positions x_c(c) to cell positions
      x(i):
14 i ← 1;
15 for all clusters c do
16 |  x = x_c(c);
17 |  for i ≤ n_last(c) do
18 |  |  x(i) ← x;
19 |  |  x ← x + w(i);
20 |  end
21 end
```

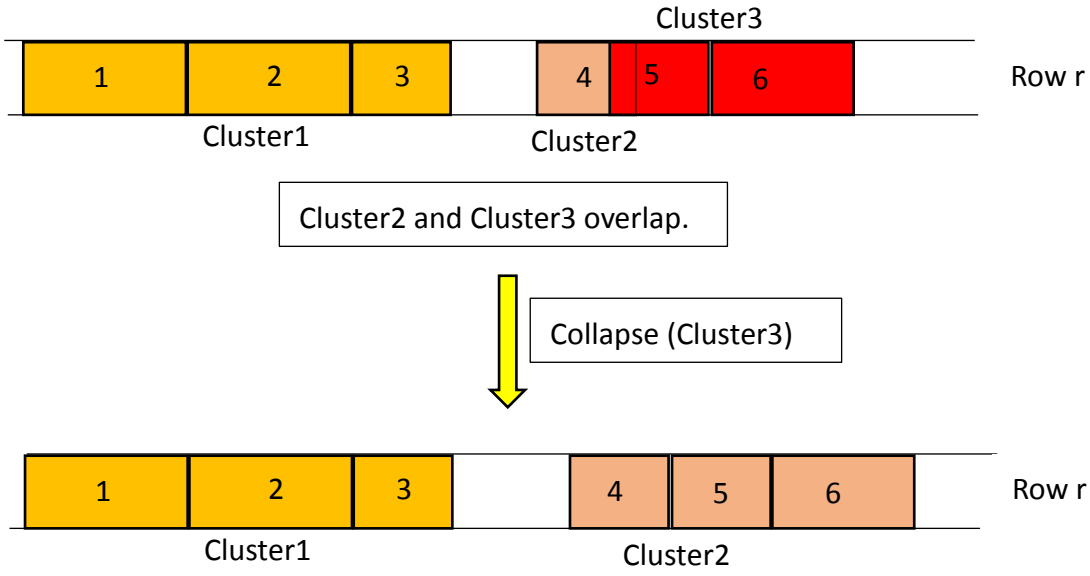
A **cluster** is defined that a set of cells that are placed compactly in x direction. Below shows an example of clusters.

In [1], a best position of a cluster c is proved to be $q(c)/e(c)$ where e(c) is the total weight of all the cell in cluster c. In this project, all cells have weight 1. Furthermore, the functions "**AddCell**" and "**AddCluster**" are shown below. They update the e, q, w, n_{last} values of a cluster.

Function AddCell(c, i):
$n_{last}(c) \leftarrow i;$
$e_c(c) \leftarrow e_c(c) + e(i);$
$q_c(c) \leftarrow q_c(c) + e(i) \cdot (x'(i) - w_c(c));$
$w_c(c) \leftarrow w_c(c) + w(i);$

Function AddCluster(c, c'):
$n_{last}(c) \leftarrow n_{last}(c');$
$e_c(c) \leftarrow e_c(c) + e_c(c');$
$q_c(c) \leftarrow q_c(c) + q_c(c') - e_c(c') \cdot w_c(c);$
$w_c(c) \leftarrow w_c(c) + w_c(c');$

Last but not least, the function "Collapse" is to cope with two overlapping clusters. An example is shown below.



- **Problem and Improvement**

1. In the experiments, I find that the function "Collapse" is called a lot of times since in PlaceRow (trial) and PlaceRow (final), it will be called. Therefore, I record the result of the clusters in all rows when executing PlaceRow (final). And when PlaceRow (trial) is called, it will not waste time to make clusters for the already placed cells. By this approach, the

executing time improved 50%. It makes sense since the number of calls of "Collapse" reduced to half of the non-improved one.

2. It is not necessary to try every row while inserting a cell. I apply branch and bound approach to reduce running time.
   If

$$|y(r) - y(i)| \geq c_{best},$$
$$y(r): lower\ y - coordinate\ of\ row\ r,$$
$$y(i): lower\ y - coordinate\ of\ cell\ i$$

   , we can skip to insert cell i into row r since the cost of inserting cell i to row r will never be less than c_{best}. By this approach, the executing time improved approximately 5%.

- **Reference**

[1] Peter Spindler, Ulf Schlichtmann, and Frank M. Johannes, "Abacus: Fast legalization of standard cell circuits with minimal movement," Proc. ISPD, pp. 47-53, 2008.