- **Algorithm Flow**

Simulated Annealing (Fast SA[2]):

1. **Solution space**

In this project, I implement **B*-tree[1]** to represent a floorplan. Therefore, all the B*-tree structures are the solution space.

2. **Neighborhood structure**

There are **<u>five</u>** modes of perturbation in this project.

(1) Swap two nodes

Randomly choose two nodes (excluding null nodes) in B*-tree, and swap them.

(2) Delete and insert

Different from traditional tree deletion and insertion, I implement this operation by introducing null nodes in B*-tree. A null node is a node with **width = 0** and **height = 0**. When a B*-tree is built, several null nodes will be inserted in the very beginning.

Randomly choose a node (excluding null nodes) and a null node in B*-tree, and **swap** them. Figure 1 shows a delete-and-insert operation of node 2.

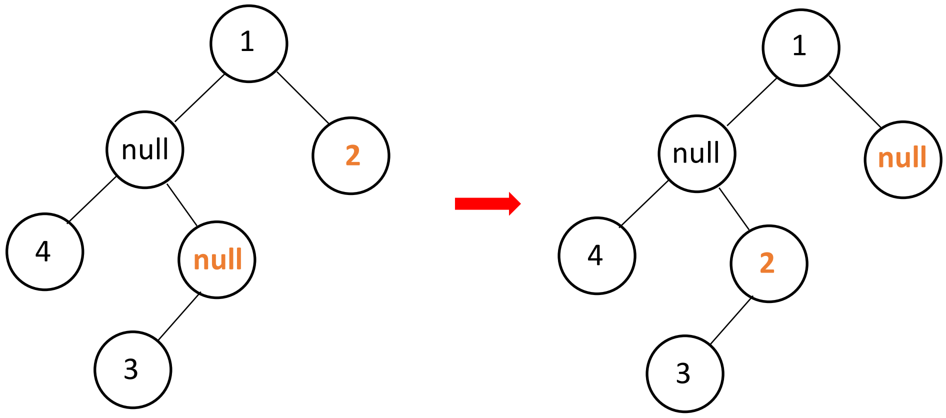

Figure 1

(3) Rotate a module

Randomly choose a node in B*-tree (excluding null nodes), and rotate it (i.e. swap its width and height).

(4) Left rotate B-tree*

Randomly choose a node in B*-tree (excluding null nodes), and left rotate the subtree rooted by the selected node. This operation may somehow increase the width and decrease the height of the floorplan.

(5) *Right rotate B*-tree*

Randomly choose a node in B*-tree (excluding null nodes), and right rotate the subtree rooted by the selected node. This operation may somehow decrease the width and increase the height of the floorplan.

3. Cost function

$$\mathrm{Cost}(\phi) = \alpha \frac{A}{A_{norm}} + \beta \frac{W}{W_{norm}} + (1 - \alpha - \beta)(R - R^{*})^{2}$$

R: The aspect ratio of the floorplan ϕ.

R^*: The aspect ratio of the given outline.

From observation of experiments, when β is small, the success rate of fitting in the fixed-outlined region can be improve significantly.

4. Annealing schedule

```
begin
Get initial solution S
Initial perturbation    /* get A_norm, W_norm, Δavg */
NumIter ← 1
T ← T(NumIter)
while(T > T_FROZEN and Not yet time out)
   for 1 ≤ i ≤ L
      Pick a random solution S'
      Δ← Cost(S') − cost(S)
      if Δ ≤ 0 then S ← S'
      if Δ > 0 then S ← S' with probability e^(−Δ/T)
   NumIter ← NumIter + 1
   T ← T(NumIter)
return S
end
```

$$
\mathrm{T(r)} = \begin{cases} -\dfrac{\Delta avg}{\ln P} & ,r = 1 \\ \dfrac{T(1) * \langle \Delta_{cost} \rangle}{rc} & ,1 < r \leq k \\ \dfrac{T(1) * \langle \Delta_{cost} \rangle}{r} & ,r > k \end{cases}
$$

Δavg: Average uphill cost

$\langle \Delta_{cost} \rangle$: Average cost change since the SA started

r: Number of iterations

c, **k**: User specified parameters (In this project, $c = 100, k = 7$)

P: Initial acceptance rate (In this project, $P = 0.987$)

- **Data Structure**

1. B*-Tree

B*-Tree is implemented by a **binary tree.** Every block has their left child, right child, and parent.

2. **Contour Line**

Contour line is a tool that helps to calculate y-coordinate of a module while packing. Given an interval of x-coordinate, we can obtain the maximum y value in this interval. Besides, while packing the blocks, we need to not only get the y-coordinate from the contour line and but also update it at the same time. In my program, it is implemented by a **doubly linked list**.

```
class ContourLine
{
private:
    int                 _height;
    ContourLine*        _front;
    ContourLine*        _back;
    pair<int, int>      _interval;
};
```

Figure 2 and figure 3 illustrate an example how to maintain contour line after inserting a 15x5 block in the interval (0, 15).

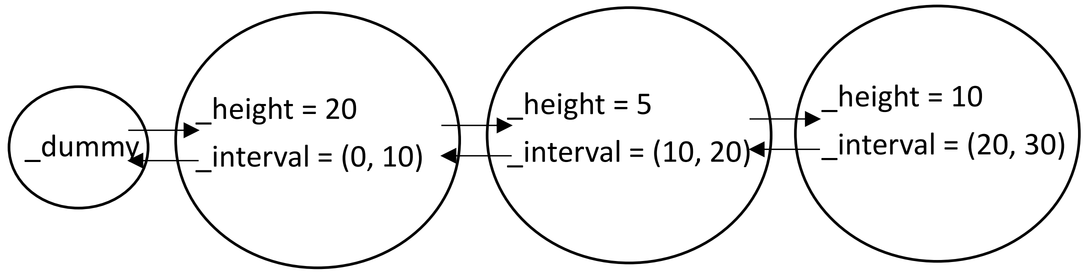

Figure 2a

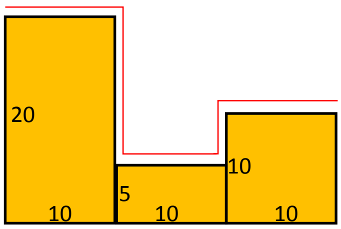

Figure 2b

Figure 2. Figure 2a shows doubly linked list which represent the contour line of the floorplan in figure 2b.

Insert a **15x5** block in the interval **(0, 15)** ……

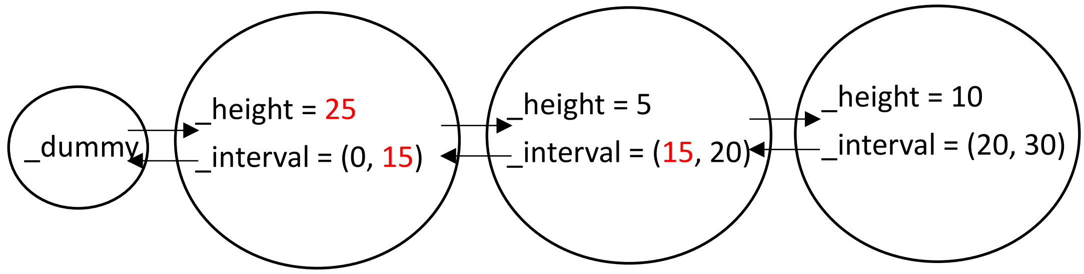

Figure 3a

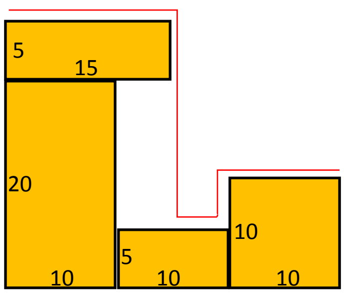

Figure 3b

Figure 3. Figure 3a shows doubly linked list which represent the contour line of the floorplan in figure 3b.

● Problem & Discussion

Q1: What if the floorplan cannot fit in the fixed-outline region for a long time?

A: In this project, if the floorplan cannot converge to fit in the fixed-outline region for more than (#Blocks × 20) iterations. It will restart the SA process (from generating an initial solution).

Q2: Sometimes, the floorplan will converge to a solution that can fit in the fixed-outline region if we rotate it (shown in Figure 4). How can we deal with it?

A: When checking if the floorplan is fit in the fixed-outline region, check both conditions — <u>rotated fit</u> or <u>non-rotated fit</u>.

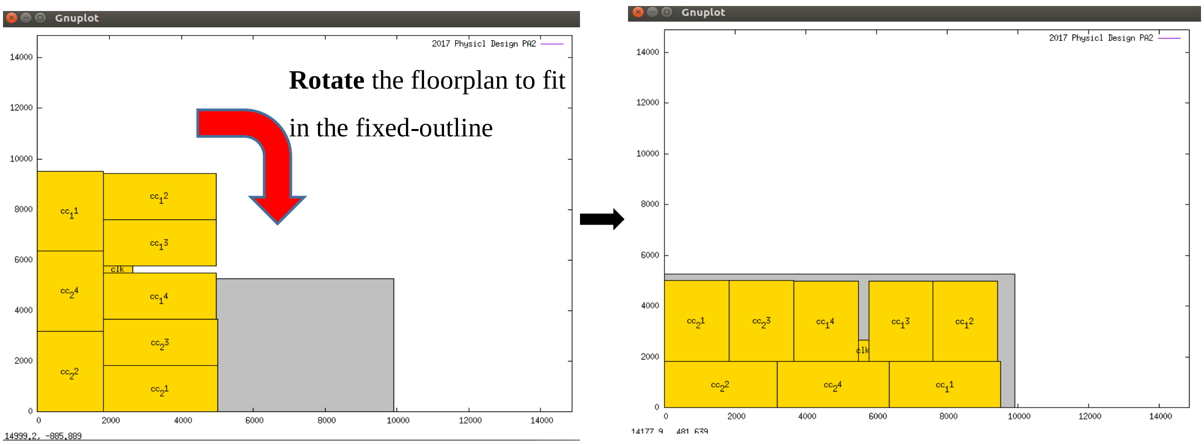

Figure 4

Q3: In the cost function, what is the possible reason that when β is small ($\beta < 0.001$), the success rate for floorplans to fit in the fixed-outline region increases?

A: In my opinion, since the most important mission is to make the floorplan fit into the fixed-outline region, the wire length should not play a vital role in the cost function. On the contrary, the aspect ratio and the area (which can help to minimize total area and thus fit in the fixed-outline region) is more important in the cost function.

● Reference

[1] Y.-C. Chang, Y.-W. Chang, G.-M. Wu, and S.-W. Wu, "B*-trees: A new representation for non-slicing floorplans," in Proc. *ACM/IEEE Design Automation Conf.*, Los Angeles, CA, Jun. 2000, pp. 458–463

[2] T.-C. Chen and Y.-W. Chang, "Modern floorplanning based on B*- trees and fast simulated annealing," *IEEE Trans. Comput.-Aided Design Integrat. Circuits Syst.*, vol. 25, no. 4, pp. 637–650, Apr. 2006.