# PD PA2 Report by R08943094 吳牧庭

## Data structure

■ Fundamental modules

Terminal: store terminal position, name and id.

Block: in addition to terminal, store block width, height, and the corresponding node on the B*-tree.

Net: store the terminals and blocks on the net.

Node: The basic element for B*-tree, each with corresponding block id, block orientation, and its link to other tree nodes. A B*-tree with nodes representing blocks and the orientation could decide a unique floorplan.

■ B*-tree

B*-tree stores tree topology and cost and supports perturbation of (1) remove & append, (2) rotate, and (3) swap.

Remove: only remove nodes with no more than 1 child.

Append: only append to nodes with no more than1 child.

Rotate: flip an arbitrary node orientation.

Swap: swap 2 arbitrary nodes.

■ Contour

Contour stores every corner on the current horizontal contour. This can update block y coordinate incrementally and reduce updating all blocks' coordinates to O(n).

Conclusion: given a b*-tree, the time complexity to get all floorplan information is O(n).

## Floorplanner

■ Initial floorplan

(1) All blocks are parsed with width no smaller than height, which makes all blocks of the same orientation.

(2) Sort all blocks by their width.

(3) Place the blocks to make the B*-tree a left-skewed tree. No block is allowed to exceed outline in x coordinate.

   a. Find the current largest block to be the root or the right child of the rightmost node.
   b. Find the next block and rotate to be the left child of the block placed in a. if it doesn't exceed current max height. If it does, rotate it back and try to fit in to be the left child.
   c. It the block doesn't fit in, get next block and do b. until all blocks are tried.
   d. If all blocks are tried, go back to a.
   e. Repeat the process until all blocks are placed.

Conclusion: the time complexity to find an initial floorplan is O(n^2).

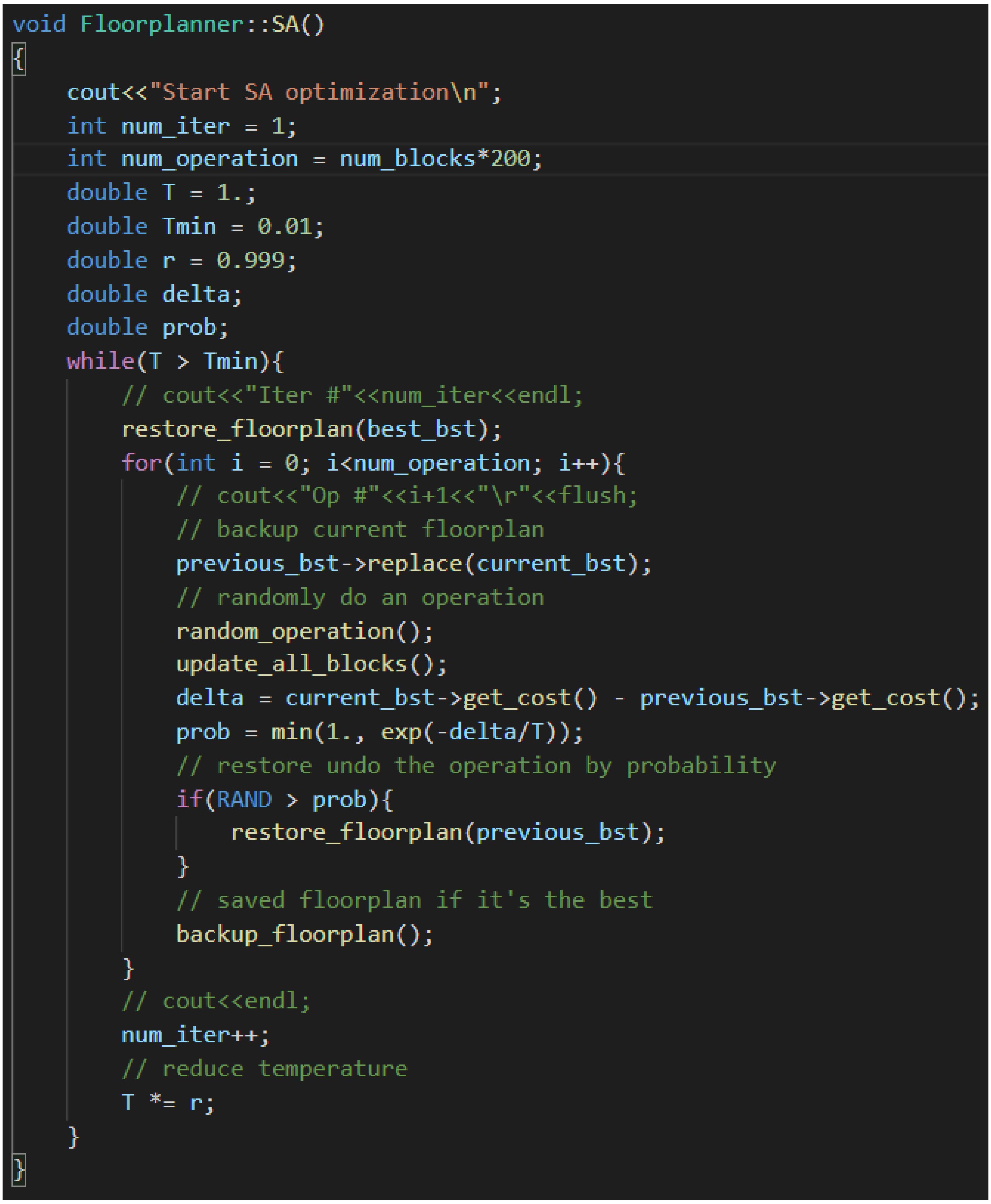
■ Simulated annealing

Solution space: all possible B*-tree topologies with al possible node orientations.

Neighborhood structure: solutions with a move away from the current solution. Moves are defined by B*-tree perturbations.

Cost function: $\alpha * \text{Area} + (1-\alpha) * \text{WL} + \lambda * (\text{outline aspect ratio} - \text{chip aspect ratio})$. Lambda is the penalty factor. If the current floorplan is illegal (blocks lying out of bound), lambda will be 10. Otherwise, lambda is 1.

Annealing schedule: See the codes below



```cpp
void Floorplanner::SA()
{
    cout<<"Start SA optimization\n";
    int num_iter = 1;
    int num_operation = num_blocks*200;
    double T = 1.;
    double Tmin = 0.01;
    double r = 0.999;
    double delta;
    double prob;
    while(T > Tmin){
        // cout<<"Iter #"<<num_iter<<endl;
        restore_floorplan(best_bst);
        for(int i = 0; i<num_operation; i++){
            // cout<<"Op #"<<i+1<<"\r"<<flush;
            // backup current floorplan
            previous_bst->replace(current_bst);
            // randomly do an operation
            random_operation();
            update_all_blocks();
            delta = current_bst->get_cost() - previous_bst->get_cost();
            prob = min(1., exp(-delta/T));
            // restore undo the operation by probability
            if(RAND > prob){
                restore_floorplan(previous_bst);
            }
            // saved floorplan if it's the best
            backup_floorplan();
        }
        // cout<<endl;
        num_iter++;
        // reduce temperature
        T *= r;
    }
}
```

The #operation per iteration is linear to the input size while the #iterations is a fixed number.

Conclusion: the time complexity of the algorithm is O(n^2).