CS6135 VLSIPDA

HW3 Fixed-outline Slicing Floorplan Designn

110061146 林汶昇

https://github.com/patata0717/CS6135-VLSI-physical-design-automation/HW3

1. Introduction

In this homework, we will implement "A New Algorithm for Floorplan Design", DAC-86, which utilize Normalized Polish expression(NPE) and Simulated Annealing(SA). We will use this method to solve Fixed-outline Floorplanning.

For my experience in HW2, I decided to write report before implementation, which will record my problem-solving process, and avoid to be too rush to write report before deadline.

2. Problem description

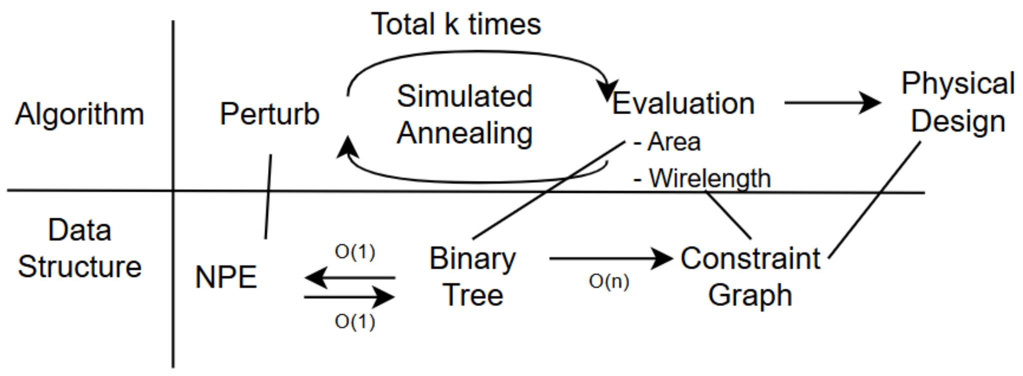
- Fixed outline: Blocks will be put on a die with width and height already known.

Input: Die size, pad locations, hardblocks(can't change shape, can rotate), nets.

Output: Hardblocks and their x, y coordinates.

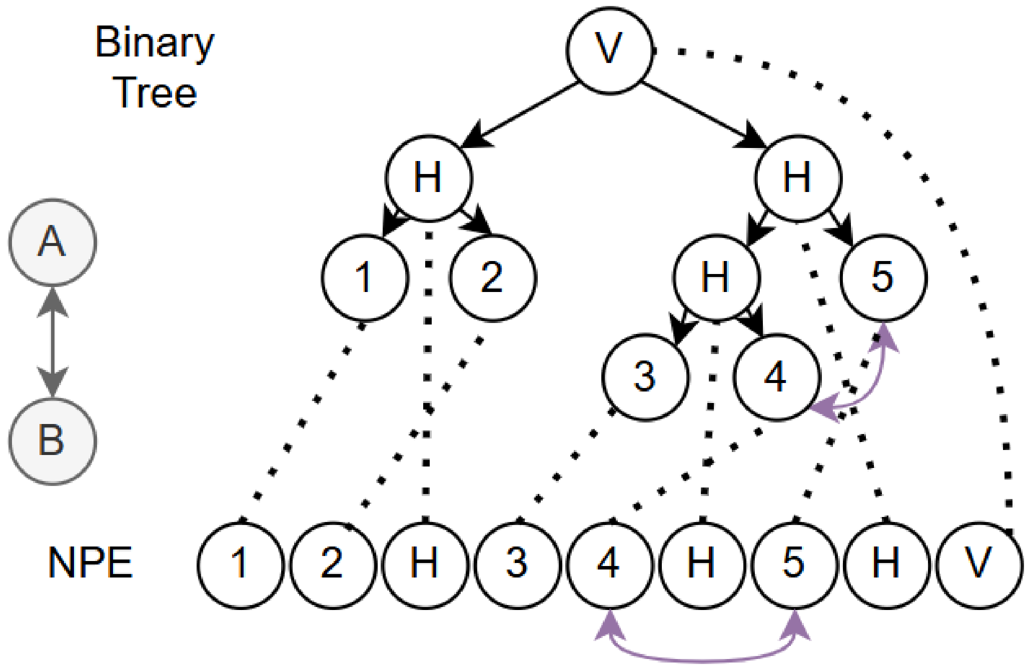
Objective: Minimize Half-Parameter-Wire-Length

3. Algorithm and Data structures

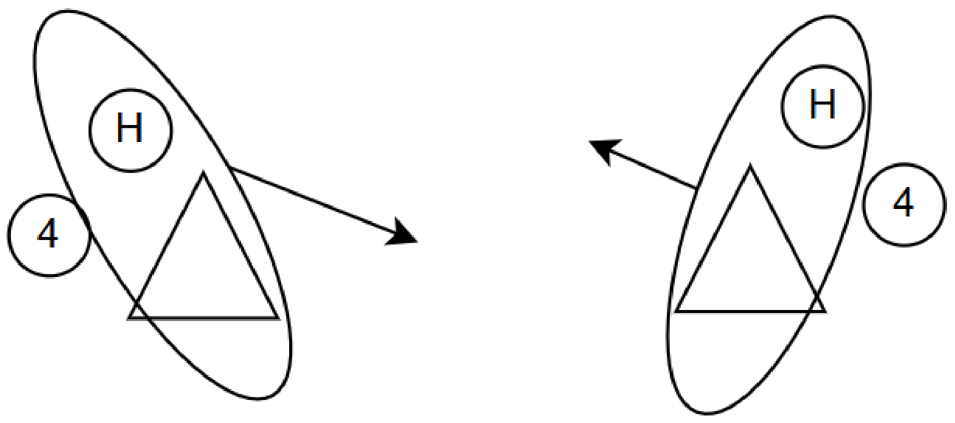


Simulated annealing is a Perturb↔Evaluation process, controlled by temperature. We perform perturb in NPE, and evaluate area in Binary Tree, evaluate wirelength in Constraint Graph, and finally we export constraint graph to physical design.

For conversion between data structure, NPE is traversal of Binary Tree, Binary Tree can be built from NPE using a Stack. Both can be done in linear time. But, if we did not discard the data for both data structure, I claim that it can be done in constant time for all Op. 1, Op. 2, and Op. 3.



We link all nodes in Binary Tree to NPE, a change in NPE can be applied on Binary Tree in constant time. For Op. 1, operand swapping(purple), with direct access to node 4, node 5, we can swich node in the tree. For Op. 2, chain invert, we can invert chain on Binary tree. For Op. 3, operator/operand swap, is more complex.



It involves bringing a node and its left/right subtree, replace a branch of other's. But, it still can be done in few steps(constant time), if we using pointer to build Binary tree rather than array.

For Binary Tree to Constraint Graph, it uses recursive call from root to leaf, and it is certainly O(n). Even a tiny change on leaf needs update all the blocks' coordinates in Constraint Graph.

So, for Perturb↔Evaluation process, it Start from random Purterb on NPE, convert to Binary tree, to Constraint Graph, and calculate each net's HPWL, so it is O(1)+O(1)+O(blocks)+O(nets)*Max_pin. So it is linear for number of blocks and nets. If we are doing k Perturbations, the complexity is O(k(blocks+nets)).

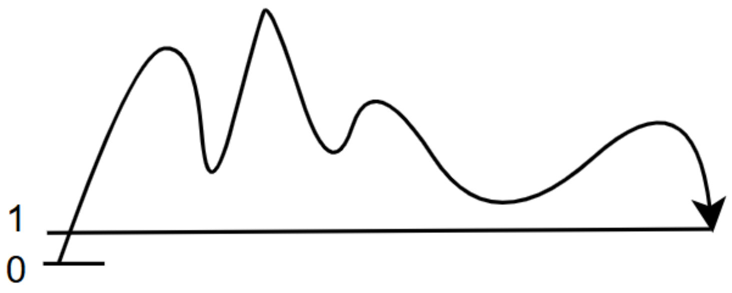
4. Invalid operation

For Op. 1 and Op. 2, all random operations are acceptable. But for Op. 3, some cases my violets the property of NPE.

- Violate balloting: In short, to convert NPE to Binary Tree, we use a Stack. A operator stands for 2 pop and 1 push, so if 2 there is no 2 elements above in the Stack, it will fails.
- Violate skewed: in the definition, same operator can't be next to each other, which means "HH", "VV" is incorrect.

So we need to do some check for generate a valid Op. 3.

Let's talk about how I maintain balloting.



$$P_1 = 25V1H374VH6V8VH$$

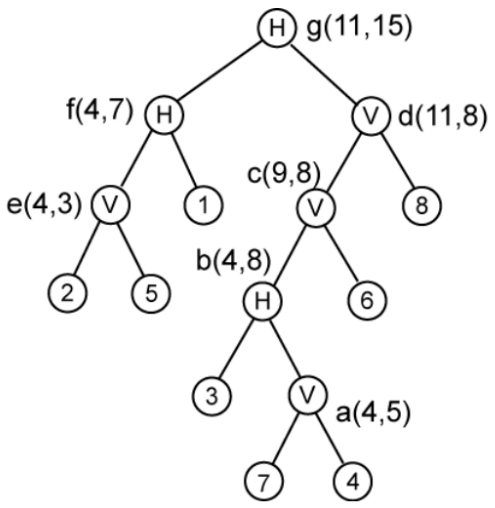
In the NPE, each operator will pop 2 operands and push a result afterward, so to avoid balloting, we need to make the number of operands always > number of operator + 1. So I will keep an array to monitor (operand – operator) value for each position on NPE, to make all positions(expect first pos) on NPE should not be less than 1. Thus we can avoid Op. 3 balloting.

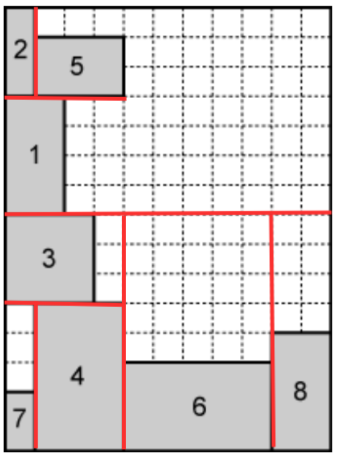
So we implement the initial data structure which turn treenodes to NPE and binary tree.

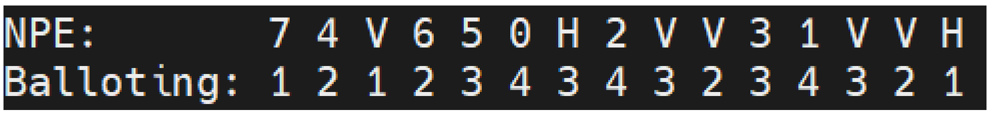
Sample1.txt:

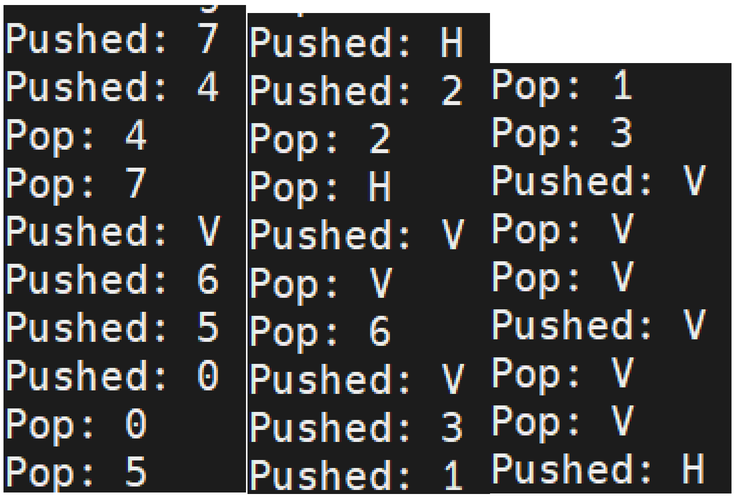
- Dimensions: (2,4), (1,3), (3,3), (3,5), (3,2), (5,3), (1,2), (2,4)

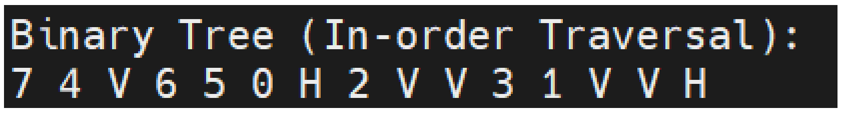
14V0H263VH5V7VH






```
NPE:       7 4 V 6 5 0 H 2 V V 3 1 V V H
Balloting: 1 2 1 2 3 4 3 4 3 2 3 4 3 2 1
```


```
Pushed: 7   Pushed: H
Pushed: 4   Pushed: 2   Pop: 1
Pop: 4      Pop: 2      Pop: 3
Pop: 7      Pop: H      Pushed: V
Pushed: V   Pushed: V   Pop: V
Pushed: 6   Pop: V      Pop: V
Pushed: 5   Pop: 6      Pushed: V
Pushed: 0   Pushed: V   Pop: V
Pop: 0      Pushed: 3   Pop: V
Pop: 5      Pushed: 1   Pushed: H
```


```
Binary Tree (In-order Traversal):
7 4 V 6 5 0 H 2 V V 3 1 V V H
```

The implementation for building NPE and Binary Tree with aware of balloting success. The traversal of Binary Tree is same as NPE.

5. Compound block

- Compound block: which means it is a combination of 2 block(either hardblock or another compound block) of a non-leaf node.

We use compound block to implement Stockmeyer's method, for efficiently build constraint graph and for selecting rotation.

If a compound block contains 2 hardblocks, each hardblock can be rotated, so there is total 2 by 2 = 4 combinations, but only 2 are optimal, which means there's no other combination can put completely inside it.

For example, A(2, 3), B(4, 5), horizontal, we have:

(2, 3) H (4, 5) = (6, 5);

(3, 2) H (4, 5) = (7, 5);

(2, 3) H (5, 4) = (7, 4);

(3, 2) H (5, 4) = (8, 6);

(6, 5) can put in to (7, 5), eliminate (7, 5)

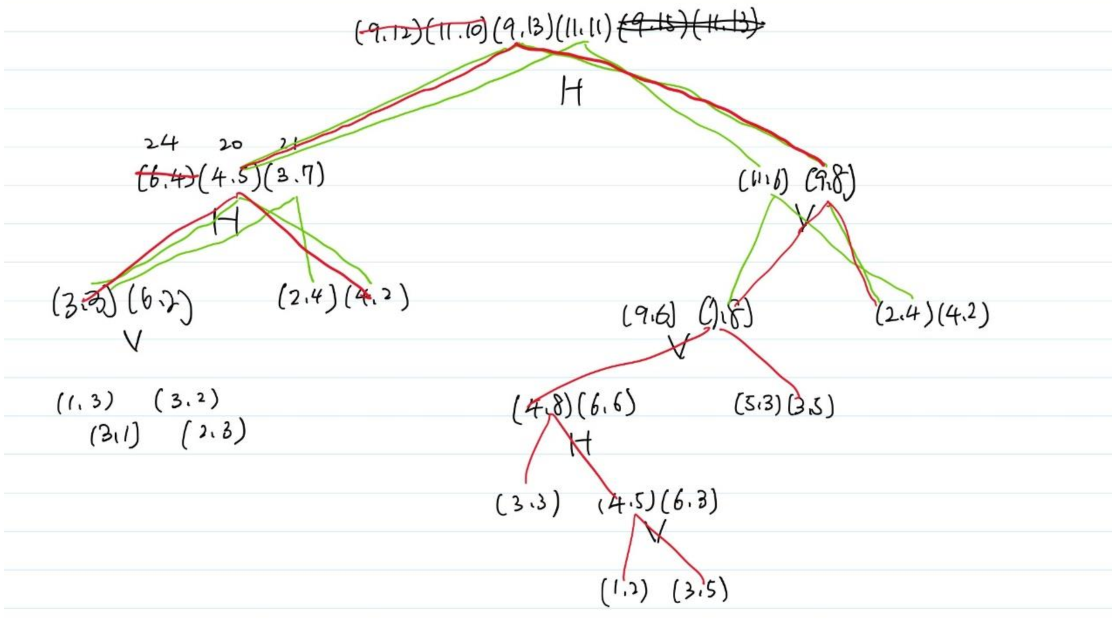
(6, 5) can put in to (8, 6), eliminate (8, 6)

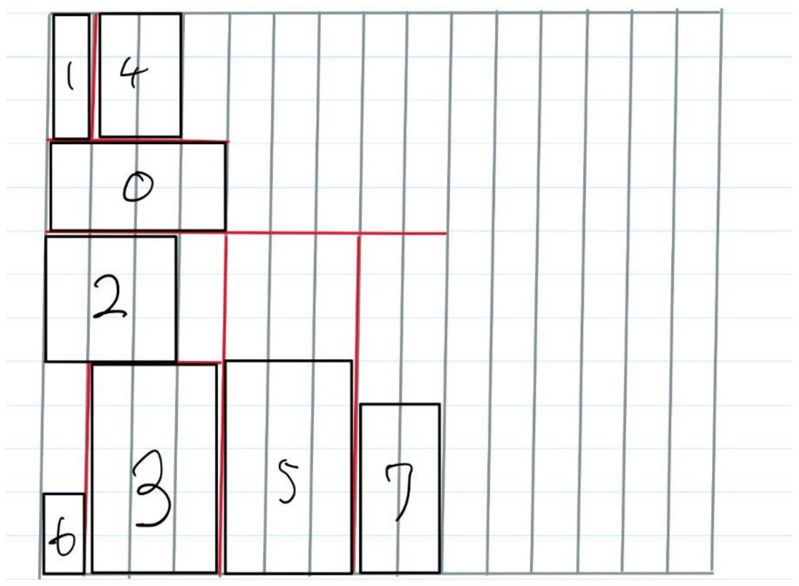
So only (6, 5), (7, 4) is optimal.

Stockmeyer approach, number of combinations in a node grows linearly with height, for height=2, it is combinations of 2 hardblocks which can rotate, certainly has 2 optimal combinations. For height=3, it may have at most 3 nodes. But in my approach, I only keep 2 best optimal node with best area(w*h) for easier implementation. Thus, this approach may not find the optimal combination(the below example will demonstrate that).

With keeping compound block, we can build constraint graph easier, using virtual block to get the location of each block, this will be done in linear time.

6. BinayTree to Constraint graph

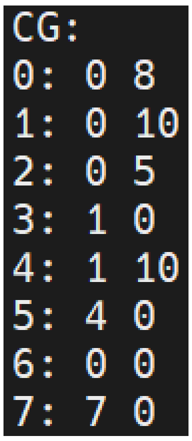




The steps for Tree2CG:

1. If it is hardblock, update its coordinates.
2. If it is compound block, see its left child and right child, tell them which combination is choosed, calculate their base coordinates, and give them the base coordinate where their (0, 0) should located.
3. Do 1, 2 recursively

Result:



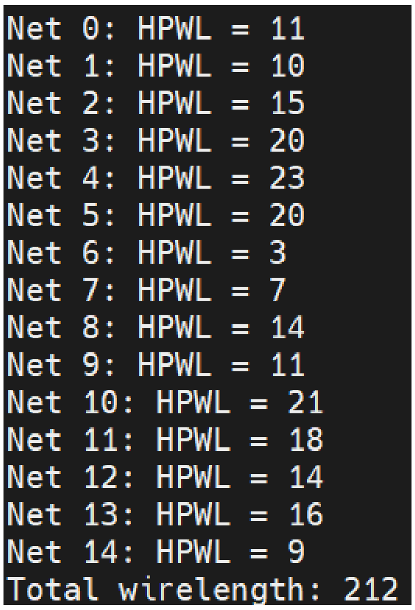
```
CG:
0: 0 8
1: 0 10
2: 0 5
3: 1 0
4: 1 10
5: 4 0
6: 0 0
7: 7 0
```

Now we finish from Initial floorplan to constraint graph.

7. Wirelength extraction

We using HPWL, which calculate the smallest bounding box of each hypernet. We set min x, min y, max x, max y, and scan through all connected pin or hardblock, and update max and min x y, and finally calculate the half parameter of the net.

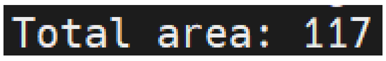
Result:



```
Net 0: HPWL = 11
Net 1: HPWL = 10
Net 2: HPWL = 15
Net 3: HPWL = 20
Net 4: HPWL = 23
Net 5: HPWL = 20
Net 6: HPWL = 3
Net 7: HPWL = 7
Net 8: HPWL = 14
Net 9: HPWL = 11
Net 10: HPWL = 21
Net 11: HPWL = 18
Net 12: HPWL = 14
Net 13: HPWL = 16
Net 14: HPWL = 9
Total wirelength: 212
```

8. Area extraction

Because we already have compound block, so just get the root's compound block of better area. It is a one-lined code.



```
Total area: 117
```

9. Maintain compound block

If I swap two nodes, we need to update 2 branches of binary tree to maintain the compound block. We use Heapify() to do this.

Cost for each perturb: O(log HardBlocks)

Area extraction: O(1)

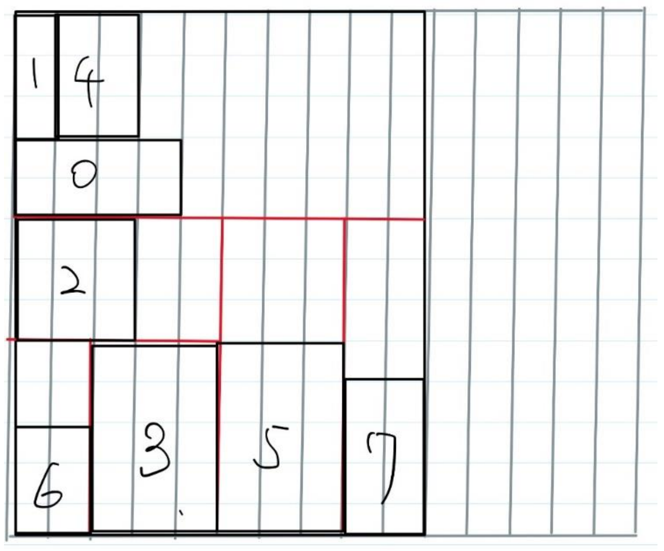
Tree2CG: O(HardBlocks)

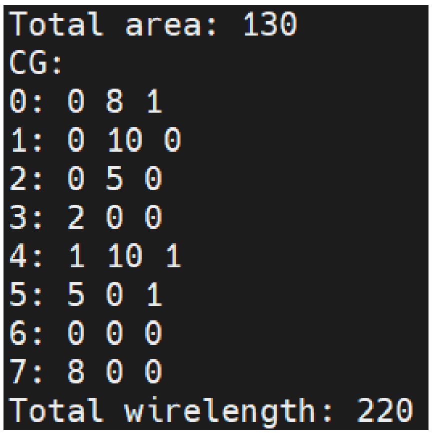
Wirelength extraction: O(Nets)

Result:

Modify block6 size from (1, 2) to (2, 3)

Heapify(block6)




```
Total area: 130
CG:
0: 0 8 1
1: 0 10 0
2: 0 5 0
3: 2 0 0
4: 1 10 1
5: 5 0 1
6: 0 0 0
7: 8 0 0
Total wirelength: 220
```

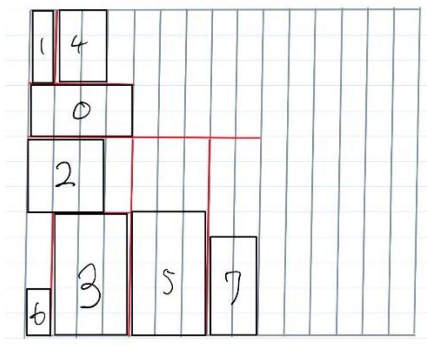
We can see that, we can get area without building CG.

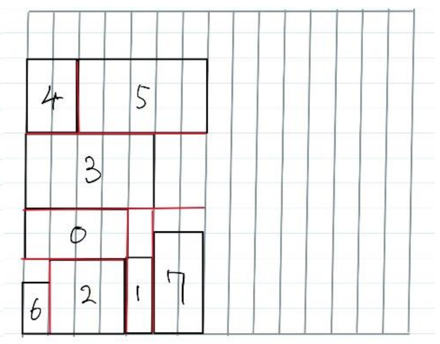
Iteration for area: O(log HB)

Iteration for wirelength: O(log HB)+O(Nets)+O(HB)

10. Perturb

Op1: Operand Swap

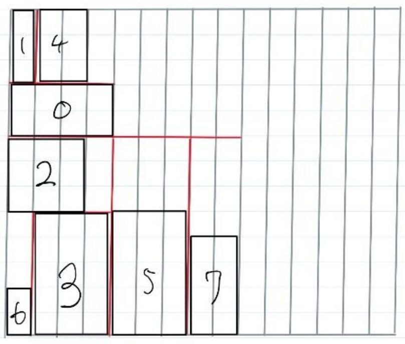


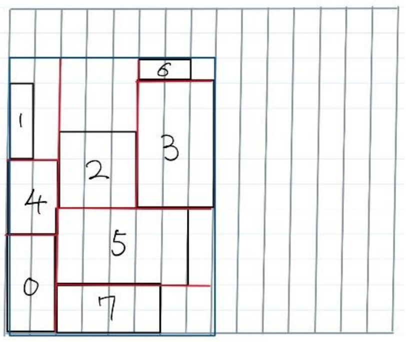


Initial(9, 13) Best(7, 11)=77

14V0H263VH5V7VH 45V3H062VH1V7VH

Op2: Chain invert

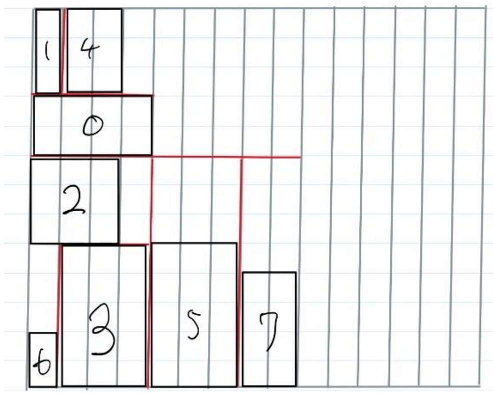


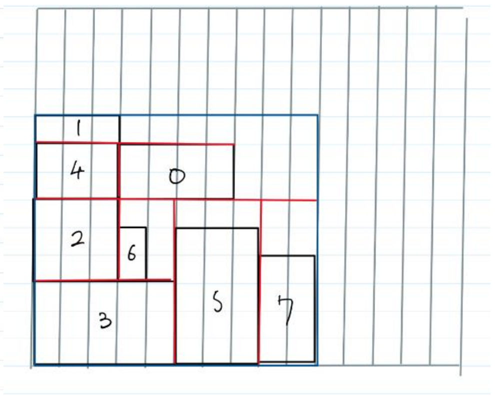


Initial(9, 13) Best(8, 11)=88

14V0H263VH5V7VH 14H0H263HV5H7HV

Op3: Operand/Operator swap

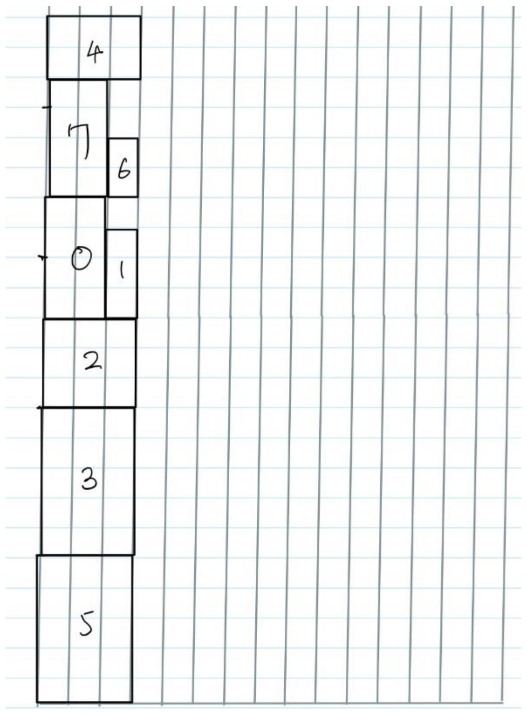




Initial(9, 13) Best(10, 9)=90

14V0H263VH5V7VH 140VH26V3H5V7VH

Combine Op1, Op2, Op3:



Best(3, 23)=69

4 7 6 V H 0 1 V H 2 H 3 H 5 H

11. Simulated annealing loop

If the result is better, accept, if the result is worse, there is a chance that it will be accept, otherwise, it will fail. The chance is based on temperature, which is a exponentially decaying number.

$$Accept\ worse\ rate = \Delta \times Temp^{(-Cooling\ Coef * Time)}$$

Delta is the distance with previous result, Ex: delta = Prev area – Curr area.
But we need to restore the previous result if it fails, so we have a backup npe, and backup binaryTree, which takes O(n) to build. So for each perturb costs O(n).

Initial: no special trick, random

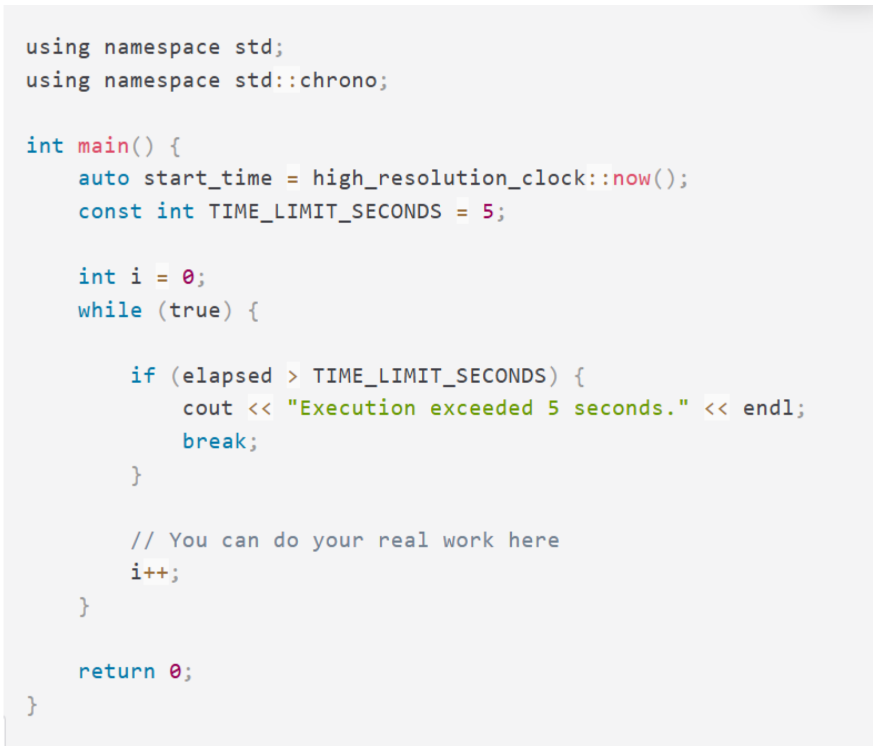
Phase 1: Area Optimization
Cost function = Minimum bounding square

Phase 2: Wirelength Optimization

Cost function = (Within Diesize) ? Wirelength : fail

12. Time Wrapper

Use a std::chrono function to wrap out simulated annealing module.

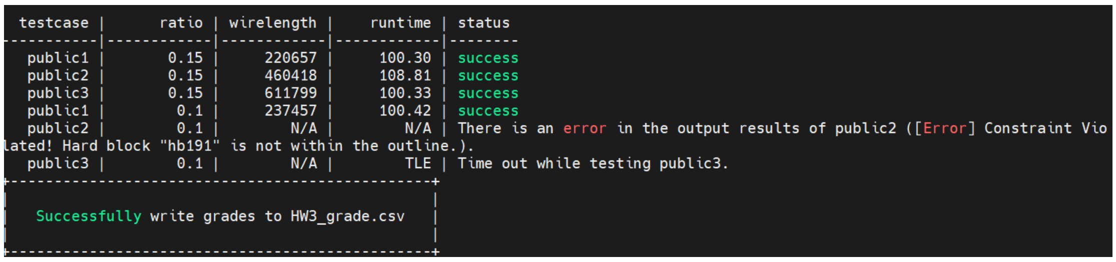


```
using namespace std;
using namespace std::chrono;

int main() {
    auto start_time = high_resolution_clock::now();
    const int TIME_LIMIT_SECONDS = 5;

    int i = 0;
    while (true) {

        if (elapsed > TIME_LIMIT_SECONDS) {
            cout << "Execution exceeded 5 seconds." << endl;
            break;
        }

        // You can do your real work here
        i++;
    }

    return 0;
}
```

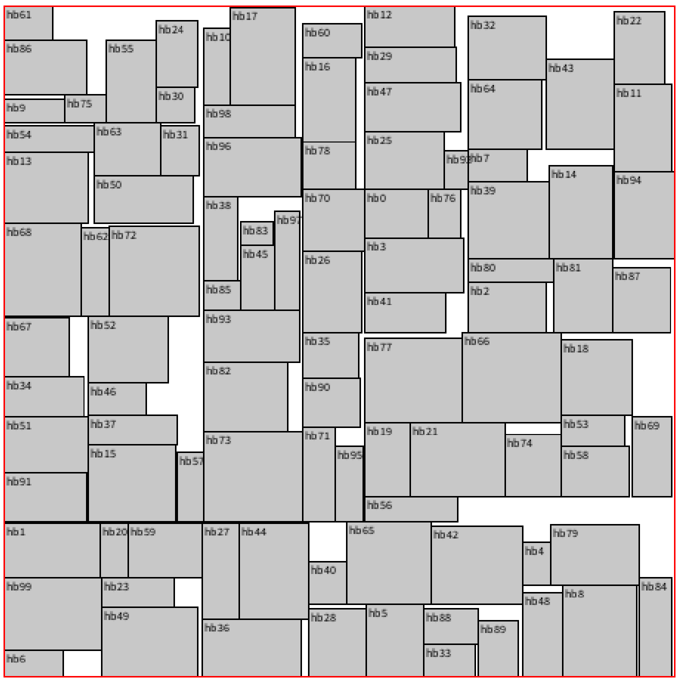
To make it end when runtime near 600 second.

13. Result

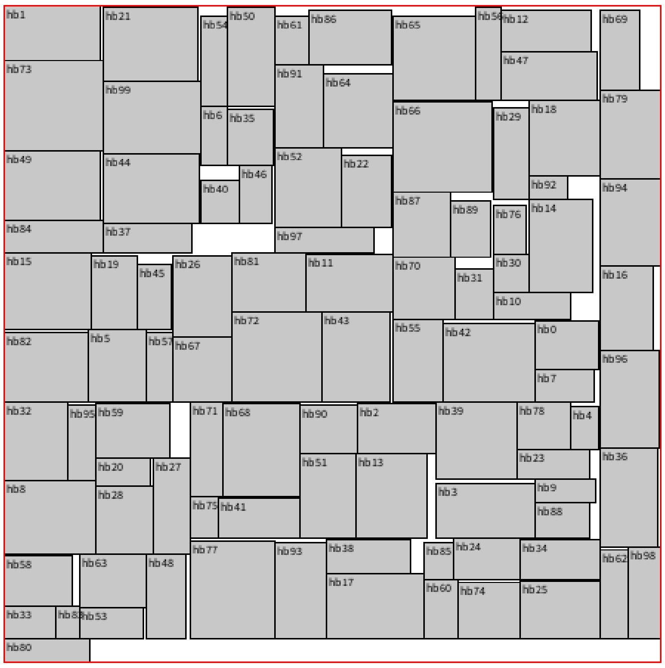
Time stg1	Public1	Stage 1	Public2	Stage 1	Public3	Stage1
0.15	454	6.43 s	477	23 s		
0.10	444	20 s	467	184 s		



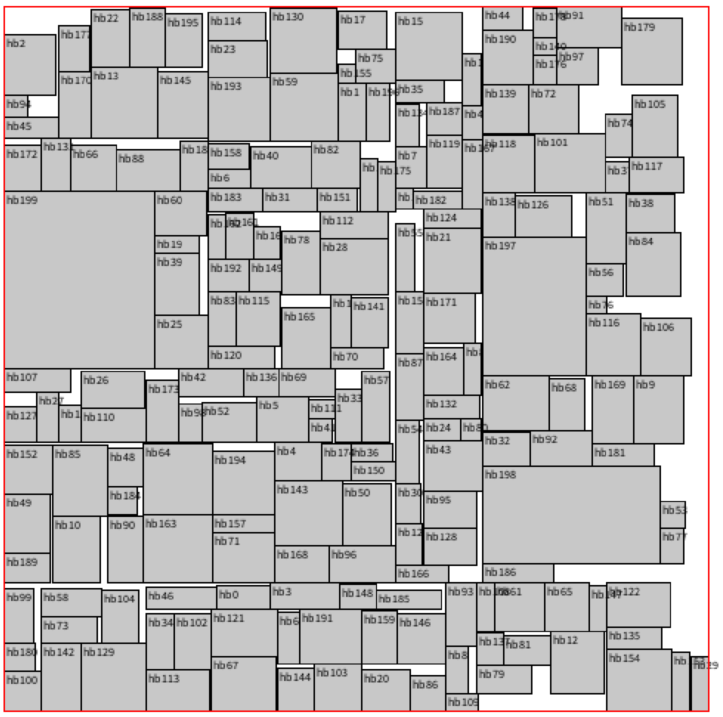
```
 testcase |       ratio | wirelength |     runtime | status
----------|------------|------------|------------|--------
  public1 |        0.15 |     220657 |      100.30 | success
  public2 |        0.15 |     460418 |      108.81 | success
  public3 |        0.15 |     611799 |      100.33 | success
  public1 |         0.1 |     237457 |      100.42 | success
  public2 |         0.1 |        N/A |         N/A | There is an error in the output results of public2 ([Error] Constraint Vio
lated! Hard block "hb191" is not within the outline.).
  public3 |         0.1 |        N/A |         TLE | Time out while testing public3.
+----------------------------------------------+
|                                              |
|   Successfully write grades to HW3_grade.csv |
|                                              |
+----------------------------------------------+
```


hb61
hb17
hb12
hb24
hb10
hb60
hb32
hb22
hb86
hb55
hb29
hb43
hb16
hb47
hb64
hb11
hb9
hb75
hb30
hb98
hb54
hb63
hb31
hb96
hb25
hb13
hb78
hb9
hb7
hb94
hb14
hb50
hb39
hb70
hb0
hb76
hb38
hb97
hb68
hb62
hb72
hb83
hb45
hb3
hb26
hb80
hb81
hb87
hb2
hb85
hb41
hb67
hb52
hb93
hb35
hb77
hb66
hb18
hb34
hb46
hb82
hb90
hb37
hb53
hb69
hb51
hb19
hb21
hb71
hb74
hb15
hb95
hb58
hb57
hb73
hb91
hb56
hb1
hb20
hb59
hb27
hb44
hb65
hb79
hb42
hb4
hb40
hb99
hb23
hb84
hb8
hb48
hb49
hb5
hb28
hb88
hb36
hb89
hb33
hb6

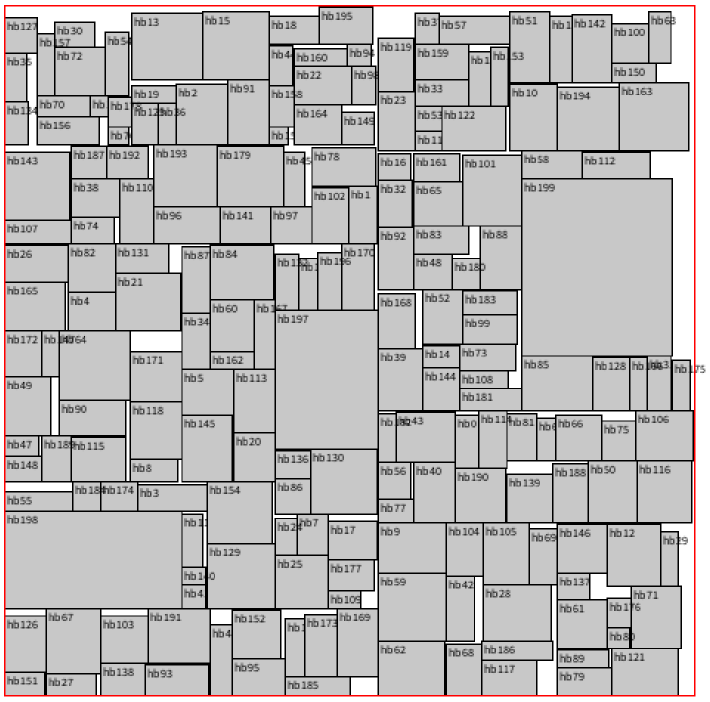
Public1 0.15


hb1
hb21
hb54
hb50
hb61
hb86
hb65
hb56
hb12
hb69
hb73
hb99
hb6
hb35
hb91
hb64
hb66
hb29
hb18
hb47
hb79
hb49
hb44
hb40
hb46
hb52
hb22
hb87
hb89
hb76
hb92
hb14
hb94
hb84
hb37
hb97
hb15
hb19
hb45
hb26
hb81
hb11
hb70
hb31
hb30
hb10
hb16
hb72
hb43
hb82
hb5
hb57
hb67
hb55
hb42
hb0
hb7
hb96
hb32
hb95
hb59
hb71
hb68
hb90
hb2
hb39
hb78
hb4
hb20
hb27
hb51
hb13
hb23
hb36
hb8
hb28
hb75
hb41
hb3
hb9
hb88
hb58
hb63
hb48
hb77
hb93
hb38
hb85
hb24
hb34
hb62
hb98
hb17
hb60
hb74
hb25
hb33
hb83
hb53
hb80

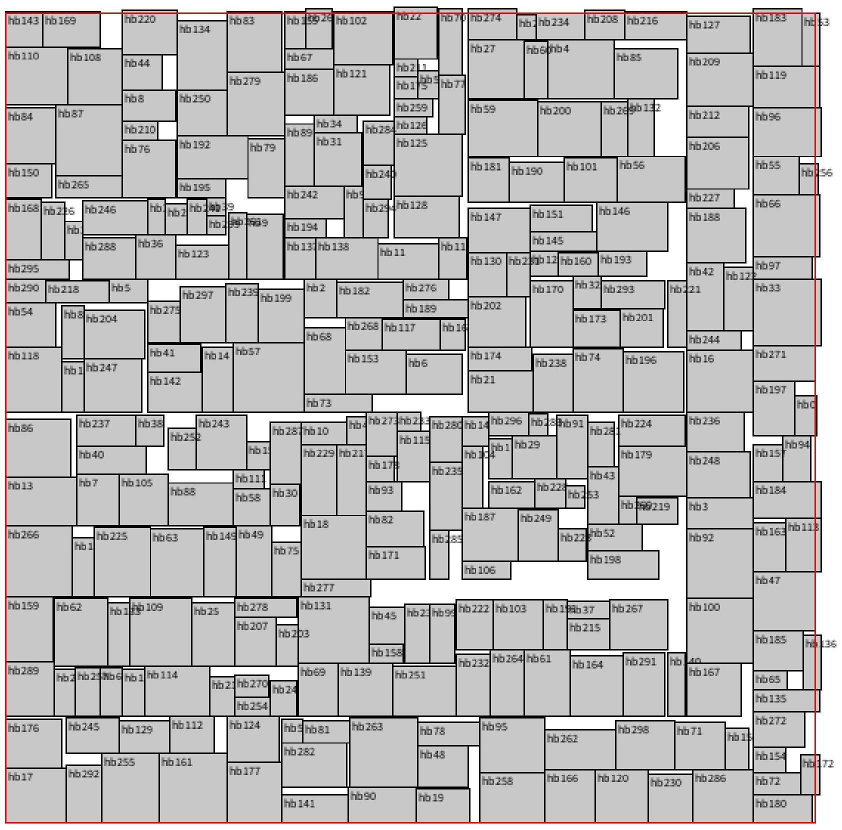
Public 0.10



Public2 0.15



Public2 0.10



Public3 0.10

- Check list

(1) Your name and student ID

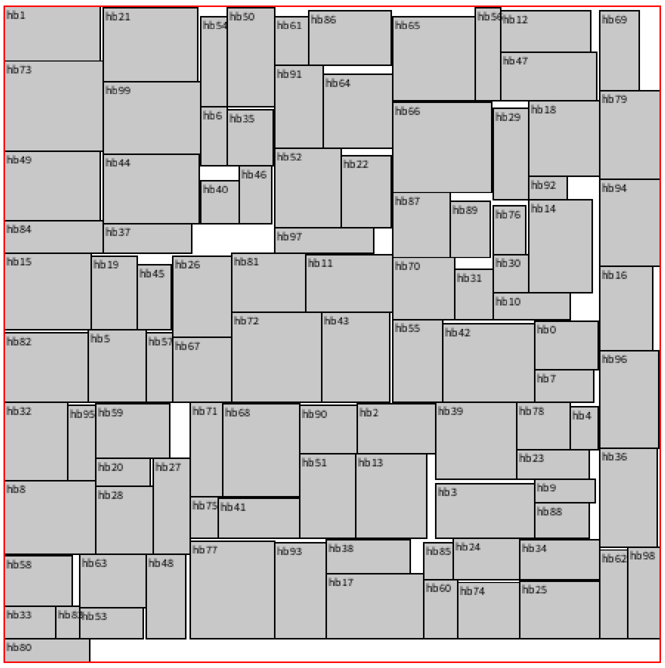
(2) How to compile and execute your program and give an execution example.

In src$ ../bin/hw3 ../testcase/public1.txt ../output/public1.out 0.15

(3) The wirelength and the runtime of each testcase with the dead space ratios 0.15 and 0.1, respectively. Paste the screenshot of the result of running the HW3_grading.sh.

(4) Please show that how small the dead space ratio could be for your program to produce a legal result in 10 minutes.

7%



(5) The details of your implementation. If there is anything different between your implementation and the algorithm in the DAC-86 paper, please reveal the difference(s) and explain the reasons.

(6) Please describe your method of your initial floorplan.

No trick, Random

(7) What tricks did you do to speed up your program or to enhance your solution

quality? Please use HW3_printer to generate the image to compare different stages (e.g., initial floorplan → trick 1 → trick 2 → final result) of your floorplanning results.

Stage 1: area optimization Stage 2: wirelenghth optimization

(8) What have you learned from this homework? What problem(s) have you encountered in this homework?

我在這個題目有兩處沒有做到最好：

1. M3 Perturb 的 Heapify 應該是能做到 O(log n)的，但是它的情況太多太複雜了，所以我還是用 O(n)去 Heapify
2. 每一個 Perturb，Backup 都要花 O(n)去 build tree，理論上可以複製一個 tree，這樣只要存取 tree 即可，但 tree 的形狀不同，要復原/複製的難度很高，所以我在這方面效率沒有做到太好。