HW3 B*Tree FloorPlan Implementation

109062526 楊皓崴

1. How to compile and execute your program, with execution example

STEP1. Access the correct directory => cd HW3/src

STEP2. Compilation => make

STEP3 Execution =>

$ ../bin/<exe> <*.hardblocks> <*.nets> <*.pl> <*.floorplan> <dead_space_ratio>

e.g.:

$ ../bin/hw3 ../testcases/n100.hardblocks ../testcases/n100.nets ../testcases/n100.pl ../output/n100.floorplan 0.1

STEP4. Verify= >

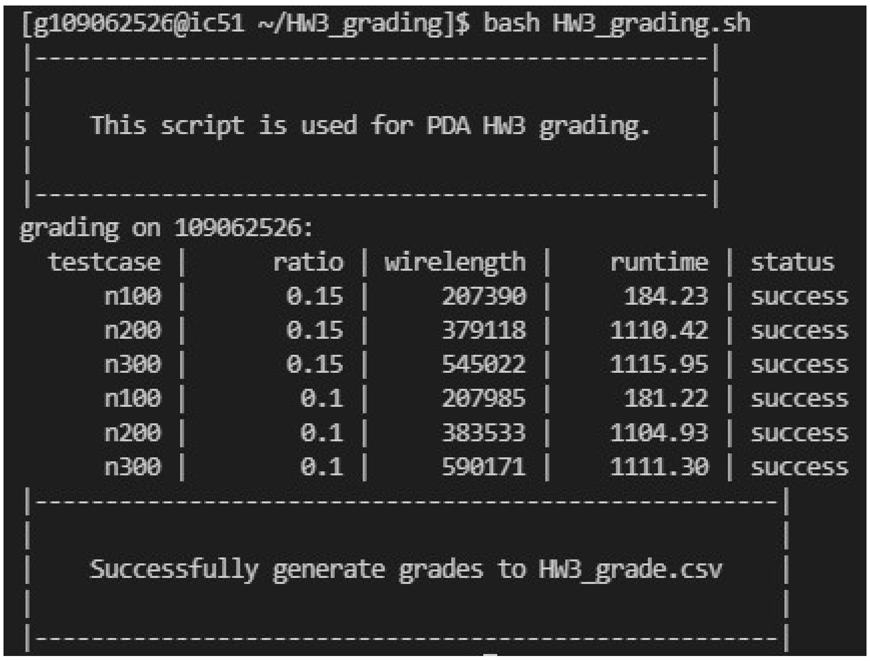
$ ../verifier/verifier <*.hardblocks> <*.nets> <*.pl> <*.floorplan> <dead_space_ratio>

e.g.:

$ ../verifier/verifier ../testcases/n100.hardblocks ../testcases/n100.nets ../testcases/n100.pl ../output/n100.floorplan 0.1

2. The wirelength and the runtime of each testcase with the dead space ratios 0.1 and 0.15

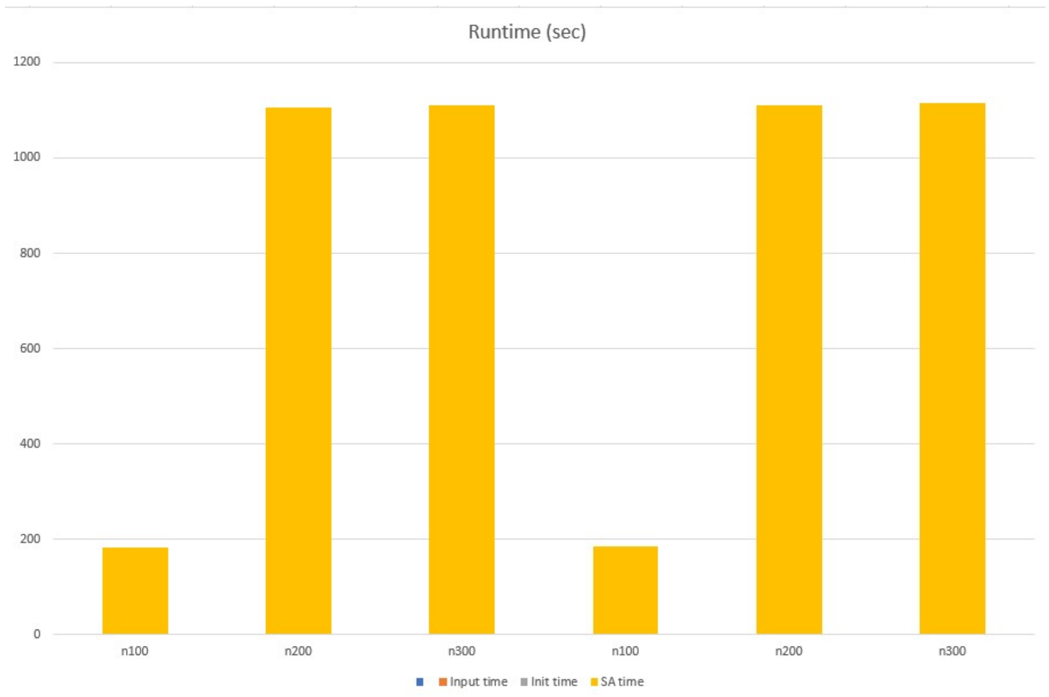
Testcase	Dead Space Ratio	Wirelength	Runtime (sec)
n100	0.1	207985	181.22
n200	0.1	383533	1104.93
n300	0.1	590171	1111.30
n100	0.15	207390	184.23
n200	0.15	379118	1110.42
n300	0.15	545022	1115.95



```
[g109062526@ic51 ~/HW3_grading]$ bash HW3_grading.sh
|--------------------------------------------------|
|                                                  |
|    This script is used for PDA HW3 grading.      |
|                                                  |
|--------------------------------------------------|
grading on 109062526:
  testcase |      ratio | wirelength |    runtime | status
      n100 |       0.15 |     207390 |     184.23 | success
      n200 |       0.15 |     379118 |    1110.42 | success
      n300 |       0.15 |     545022 |    1115.95 | success
      n100 |        0.1 |     207985 |     181.22 | success
      n200 |        0.1 |     383533 |    1104.93 | success
      n300 |        0.1 |     590171 |    1111.30 | success
|----------------------------------------------------------|
|                                                          |
|    Successfully generate grades to HW3_grade.csv         |
|                                                          |
|----------------------------------------------------------|
```

3. I/O Time Plot

Testcase	Dead Space Ratio	Input time	Init time	SA time
n100	0.1	0.00 (too small to measure)	0.00	181.22
n200	0.1	0.00	0.00	1104.93
n300	0.1	0.01	0.01	1111.30
n100	0.15	0.00	0.00	184.23
n200	0.15	0.00	0.00	1110.42
n300	0.15	0.01	0.01	1115.95



4. Please show that how small the dead space ratio could be for your program to produce a legal result in 20 minutes.

Testcase	Dead Space Ratio	Final dead space ratio
n100	0.1	0.008
n200	0.1	0.004
n300	0.1	0.01

5. The details of your algorithm.

There are 4 .cpp files in this homework.
main.cpp: the overall main file that perform this task
class.cpp: definition of the B-star tree, node, block, net
utils.cpp: methods of reading input files and output and handling trees
algo.cpp: methods of SA and init tree

- You could use flow chart(s) and/or pseudo code to help elaborate your algorithm.

pseudo code:

```
FLOORPLAN(input blockfile, netfile, pl file, deadspace){
    Init parameters, read file and store in containers
    Sort blocks by height
    Init a B-star tree "Tree" by occupying x-axis first
    //Run Simulated Annealing
    do{
        Perturb Tree
        Repack Tree
        if(cost diff < 0 or accept){
            move to new tree state
            if(improve best) record best
        }
        else(reject) move back to previous state
    }while(not TLE and not termination temperature)
    Output Tree
}
```

- If your method is similar to some previous work/papers, please cite the papers and reveal your difference(s).

Reference: Tung-Chieh Chen, *Student Member, IEEE*, and Yao-Wen Chang, *Member, IEEE*, 2006, Modern Floorplanning Based on B*-Tree and Fast Simulated Annealing

I solve this floorplan problem based on B-star Tree floorplan. The concept of left child and right child is the same in the paper and my implementation. The difference is that I did not dynamically adjust the cost function's parameter "alpha" as in "Fast SA". Secondly, I tried several kind of new cost functions, such as considering area or not, use exceeding width/height as ratio to punish invalid floorplans. Thirdly, I did not calculate Anorm and WLnorm and uphill delta_avg before SA. On the contrary, I directly adjust the parameter for a good floorplan.

Finally, and the most importantly, I did not take a three steps temperature update, since the run time is too long for my code. Generally, SA is terminated by Time limit; therefore, the temperature termination is not that critical for my SA.

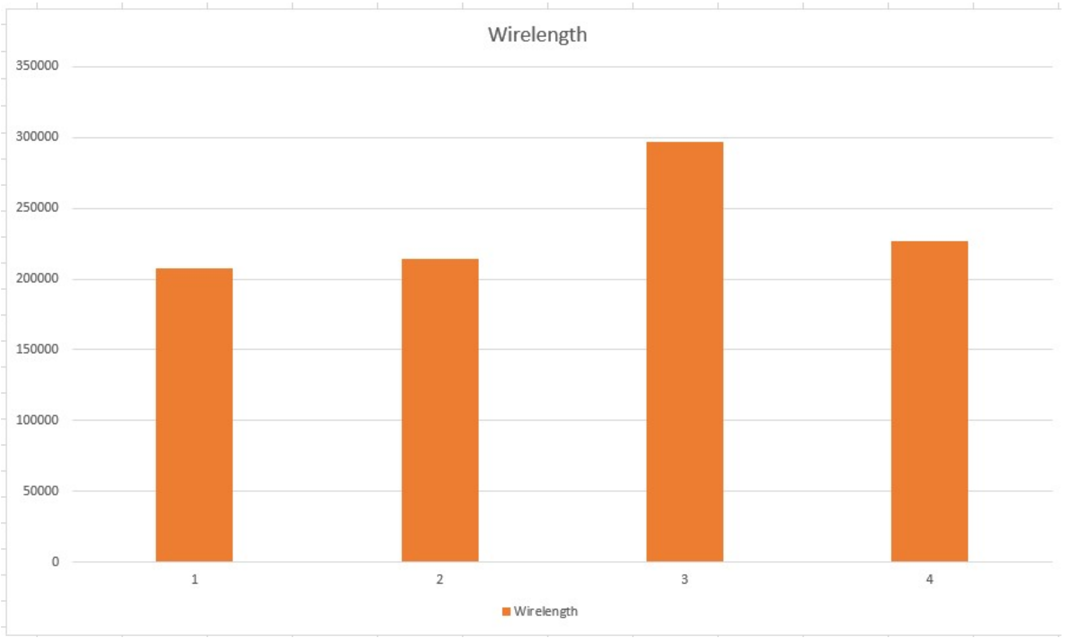
6. What tricks did you do to speed up your program or to enhance your solution quality? Also plot the effects of those different settings like the ones shown below.

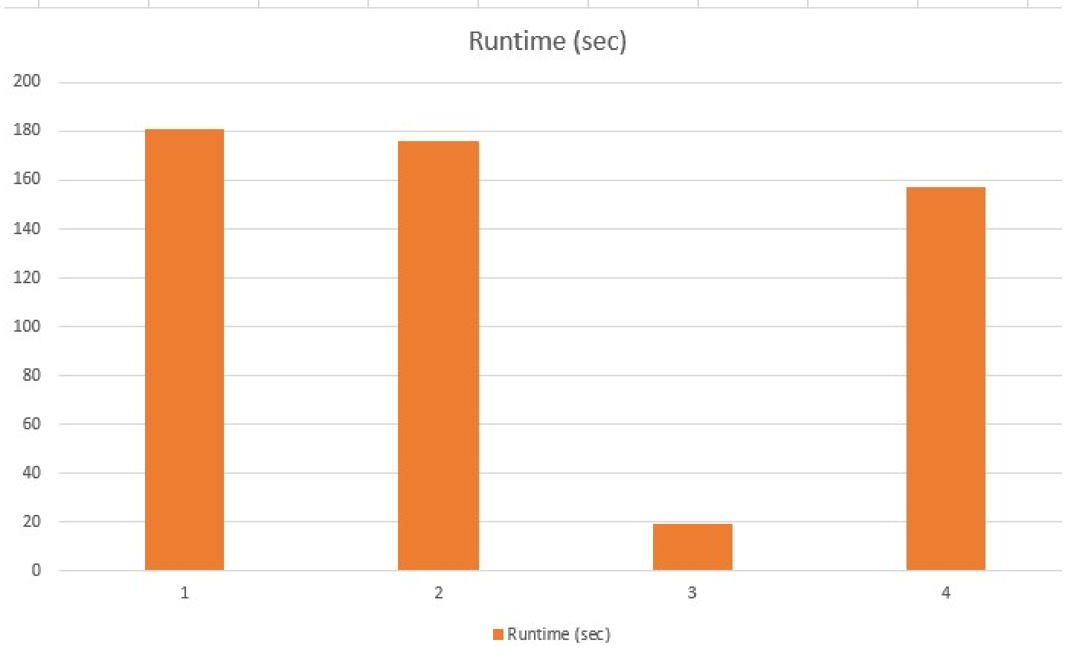
To ensure my code terminate within 20 minutes, I set a time constraint for SA. Also, I set a early stop condition to make SA stop if the best floorplan is not updated for 10 consecutive temperatures. This is the two main strategy I take. Besides, to speed up my program, I mainly focus on removing redundant code lines and redundant step in perturbing tree. However, these did not make significant improvement in runtime. I think the main reason is that in each perturbation, the actual time is really small. Even if I save time in perturbation and the perturbation is not of high quality, we still need to do a lot of perturbation in overall SA. Sometimes, we might even be stuck in local optimal. Therefore, to find a better initialized b-star tree is a better way to save runtime and achieve good result.

Different cost function:

1. 1000.0 * (ex_height + ex_width) + ((300.0 / total_blocks) * HPWL()) //adjusted scale by block num
2. 1000.0 * (ex_height + ex_width) + HPWL()
3. (ex_height + ex_width) * HPWL() //in order to 100% make sure final floorplan within outlines
4. alpha * area + (1-alpha) * HPWL()

Testcase	Dead Space Ratio	Cost Function	Wirelength	Runtime (sec)
n100	0.1	1	207985	181.22
n100	0.1	2	214276	175.89
n100	0.1	3	296595	19.31
n100	0.1	4	226976	157.22





7. Result comparison with top 5 of last year, show your advantage either in runtime or in solution quality. Are your results better?

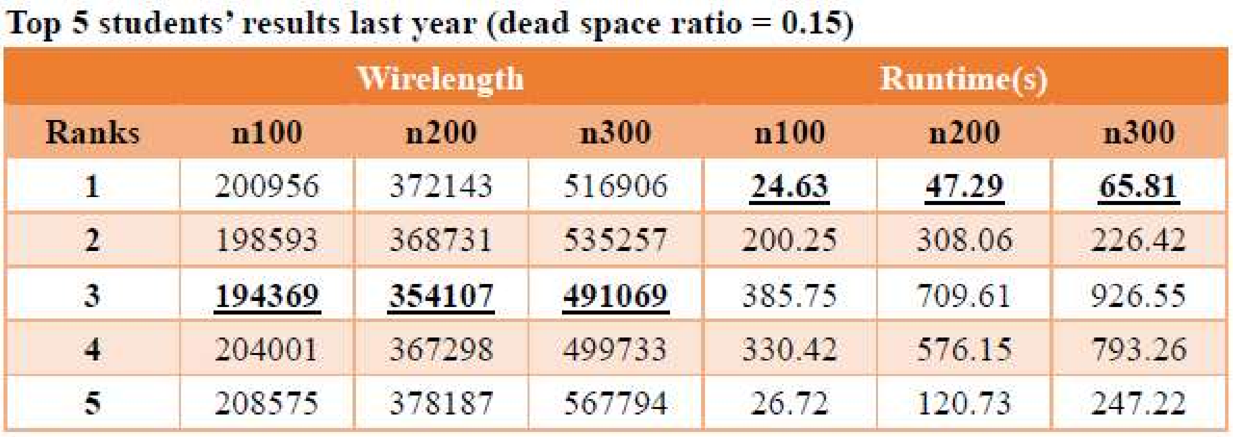
- If so, please express your advantages to beat them

I would say that my wirelength performance is within comparison range with these top5' records. Some of the wirelength is even better than some of them, such as in the n300 testcase, my wirelength is 545022. As for rank 5, its wirelength is 567794. I think my advantage to be as least as good as them in aspect of wirelength is that I have a good way to initialize my floorplan. I tried several way to initialize, and found that if I first sort the blocks with respect to their height and then place them in the manner of occupying x-axis first, the result will improve a lot. I think this is because in this manner we could save a lot of space in y-axis (height) such that the initial tree is not too high and required a lot of computation and perturbation to reduce its height.

- If not, it's fine. If your program is too slow, then what could be the bottleneck of your program? If your solution quality is inferior, what do you think that you could do to improve the result in the future?

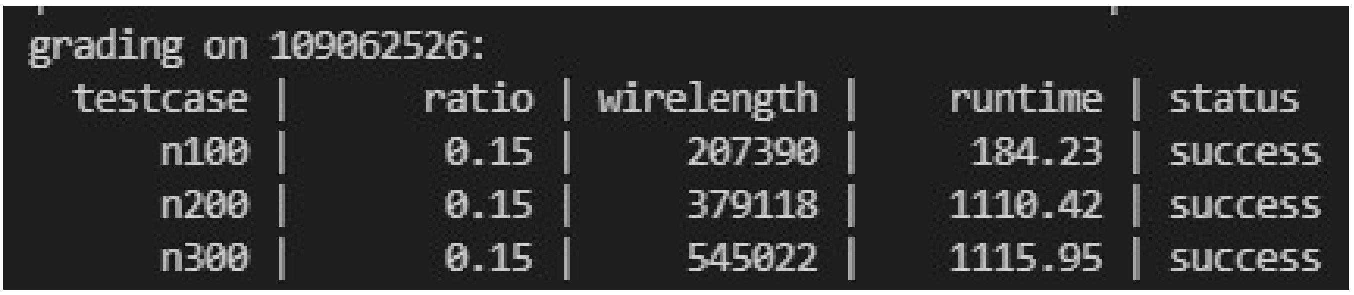
I think my runtime is definitely worse than theirs. I think the main reason is that during perturbation I need to copy a new block memory space for each new node. These require a lot of space and time. Also, I repack my tree each time I reject a perturbed tree. This is definitely redundant; however, I cannot easily delete this repack step, since I need to construct a block to node mapping relation for each original tree so that I can perturb it in the next iteration.

The way to improve this part is not to copy block for each tree. Instead, use the same set of blocks in the overall SA framework. Just maintain the rotate state of each tree since the x-coordinate and y-coordinate will only need to be compute when I need to compute the cost function. Therefore, there is no need to repack the tree in each rejection.



Top 5 students' results last year (dead space ratio = 0.15)

	Wirelength			Runtime(s)		
Ranks	**n100**	**n200**	**n300**	**n100**	**n200**	**n300**
1	200956	372143	516906	**24.63**	**47.29**	**65.81**
2	198593	368731	535257	200.25	308.06	226.42
3	**194369**	**354107**	**491069**	385.75	709.61	926.55
4	204001	367298	499733	330.42	576.15	793.26
5	208575	378187	567794	26.72	120.73	247.22

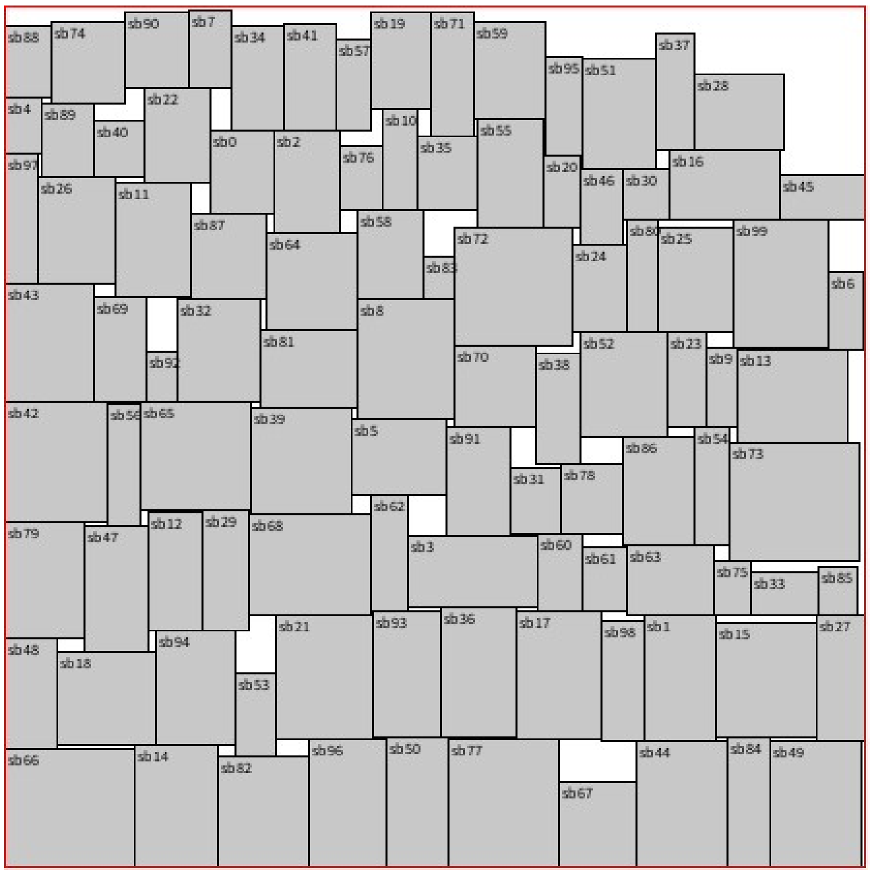


```
grading on 109062526:
  testcase |       ratio | wirelength |    runtime | status
      n100 |        0.15 |     207390 |     184.23 | success
      n200 |        0.15 |     379118 |    1110.42 | success
      n300 |        0.15 |     545022 |    1115.95 | success
```

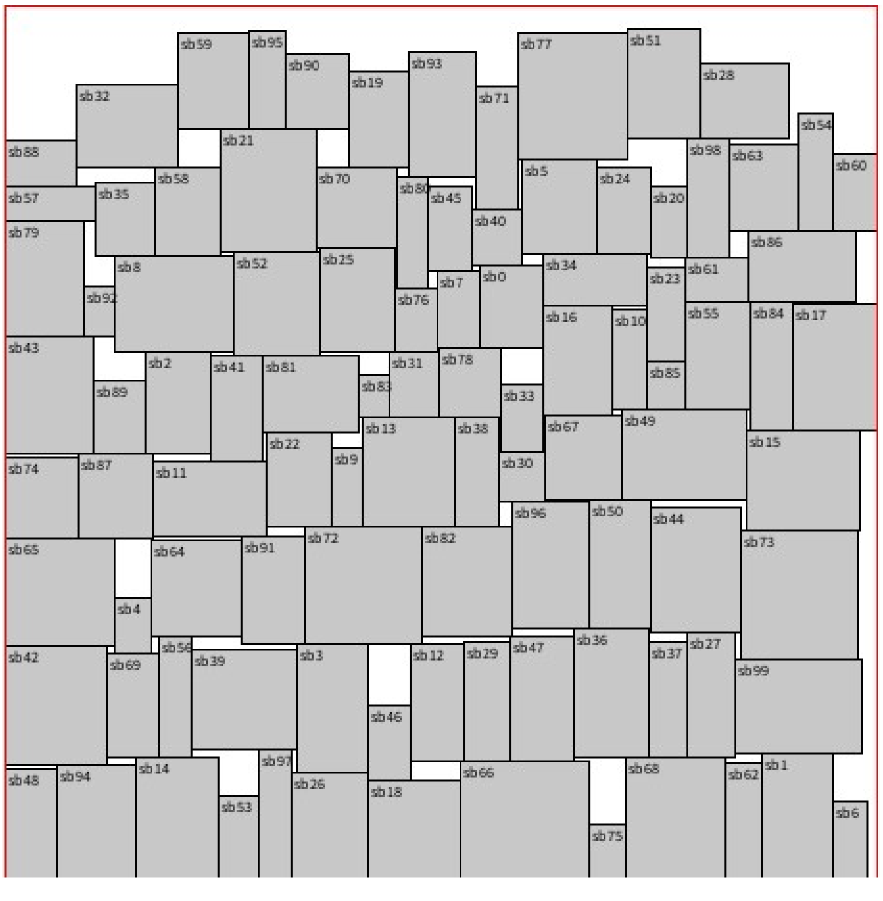
8. What have you learned from this homework? What problem have you encountered?

I think I completely review unit 4 in this course since I need to decide the algorithm to use in this homework. Thus, I need to go through the whole unit first to understand which algorithm for floorplan is better in this homework circumstance. Finally, I decided to use the Fast Simulated Annealing framework for fixed outline floorplan, since this homework is a fixed outline floorplan problem. Secondly, I reviewed a lot about the concept of DFS and BFS in implementation of this homework since building and repacking a b-star tree required to DFS the tree.

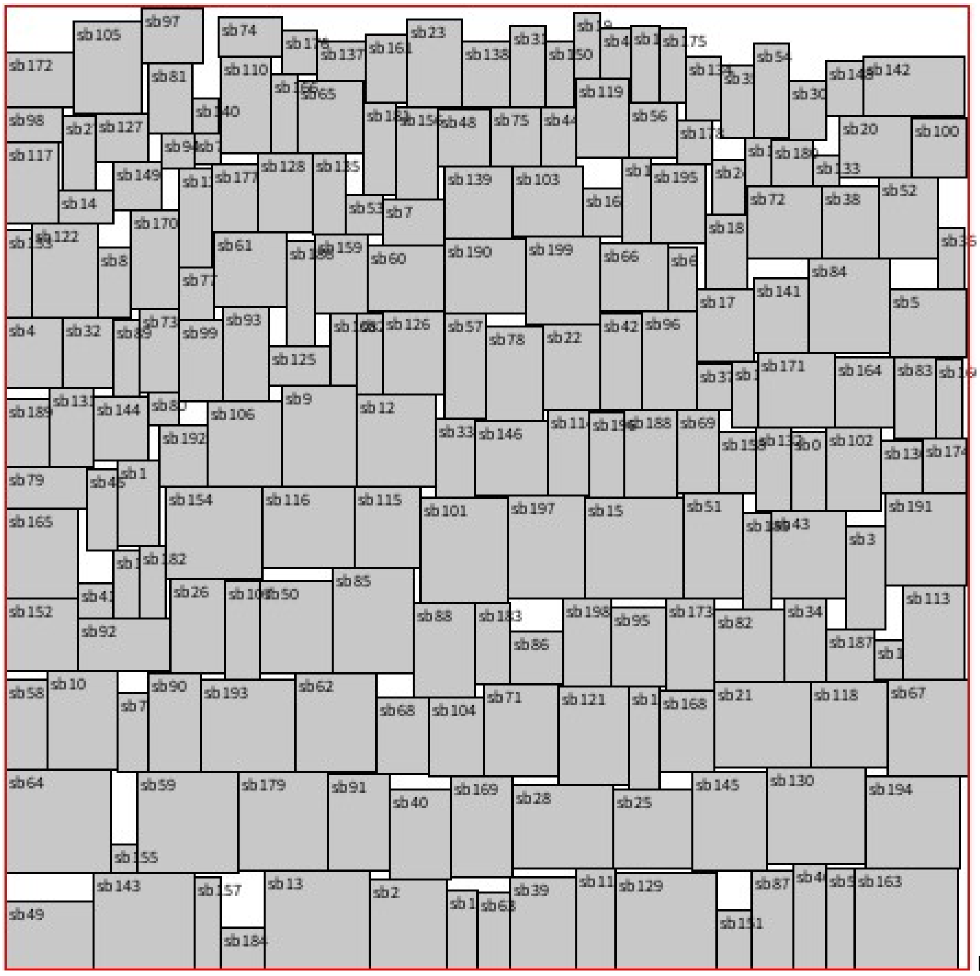
The problem I encounter is mainly about debugging perturbation methods. Since there is a lot of pointer manipulations in perturbing trees. It is very prone to encounter memory leak or infinite loops condition. For example, in “Move”, if I did not check to “to” node and the target node’s relation. It is very likely that the to node and the target node is the same node. This is leading to child of a node is pointing to itself. And causing infinite loop in DFS.

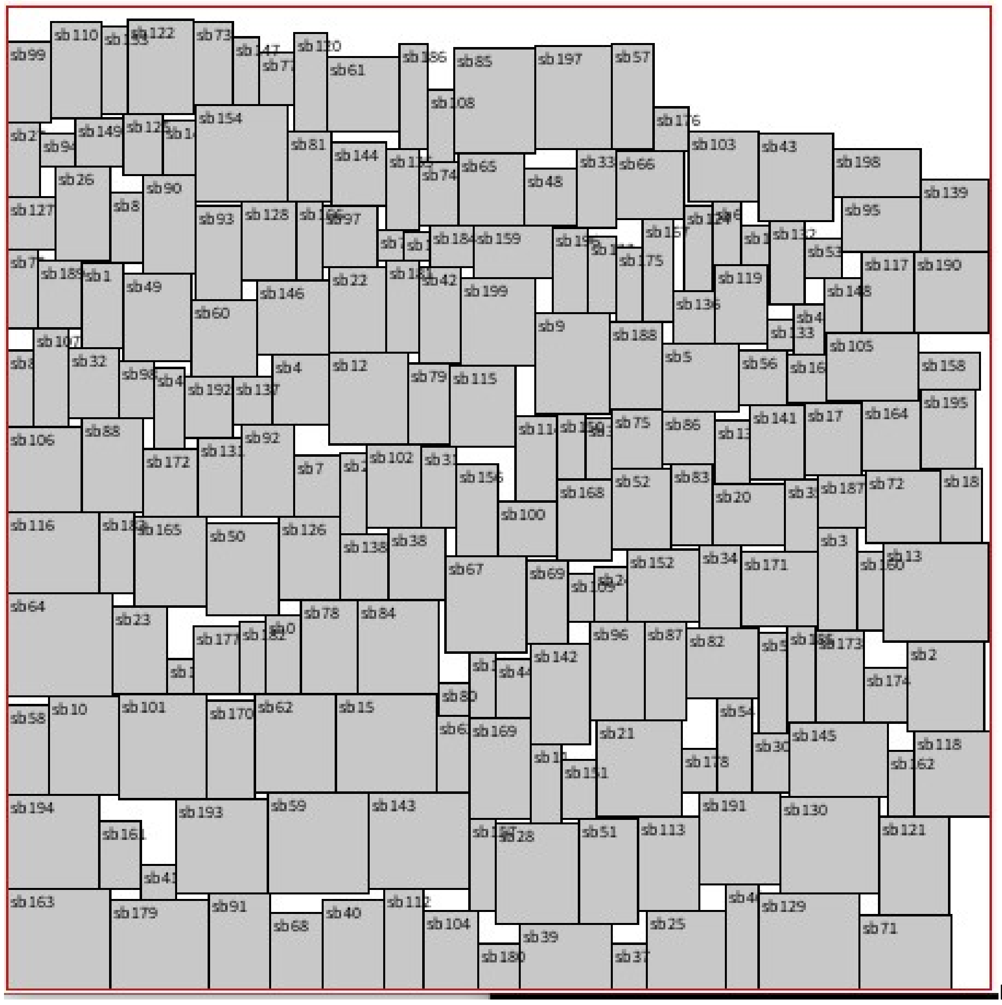


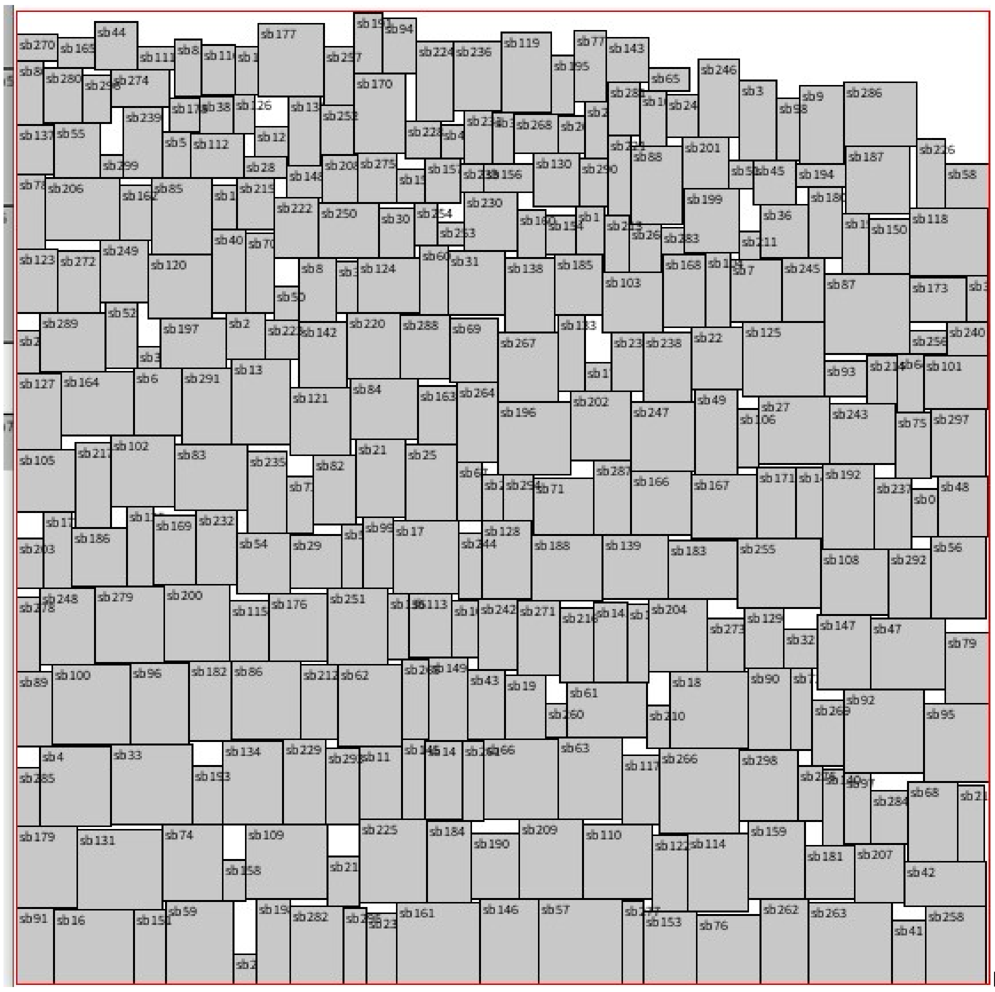
n100 / 0.1

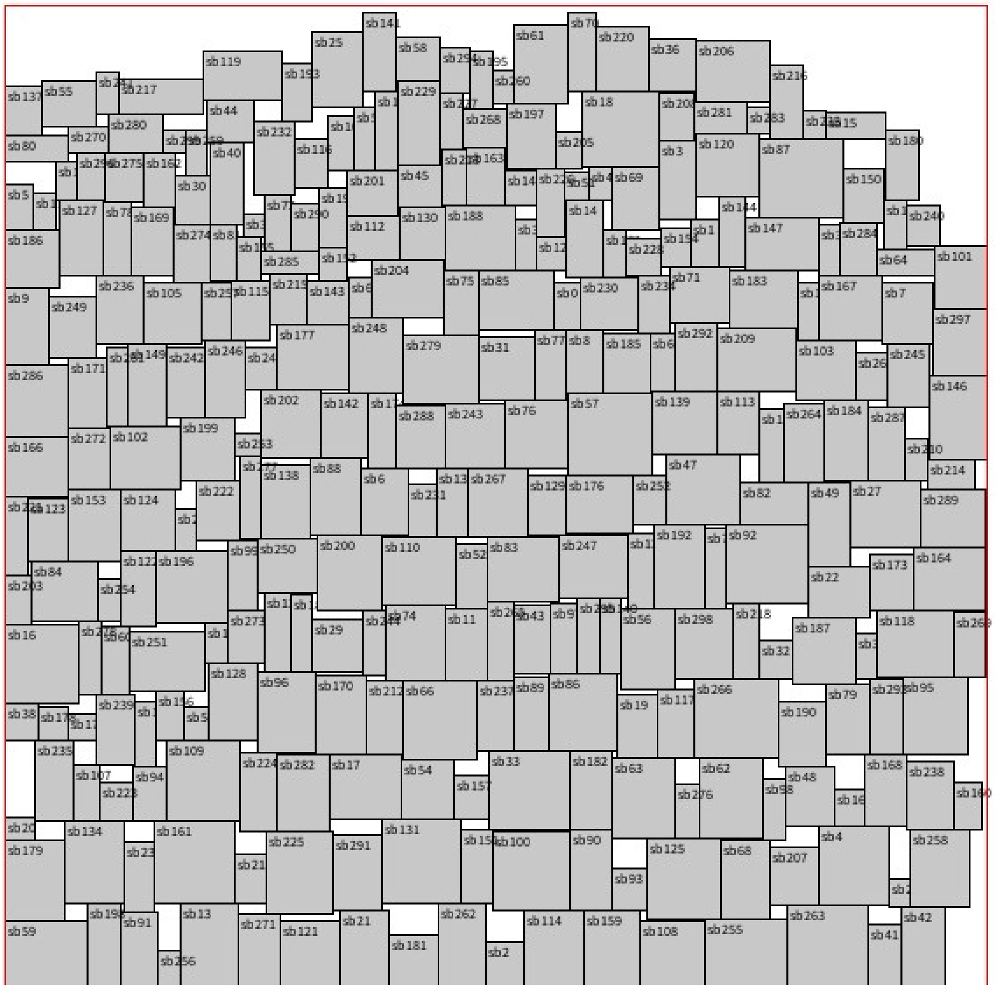


n100 / 0.15


n200 / 0.1


n200 / 0.15


n300 / 0.1


n300 / 0.15