PD PA2 Report

Programming Asignment #2 of Physical Design for Nanometer ICs, Spring 2022

系級	學號	姓名	日期
電子所碩一	r10943109	Shiuan-Yun Ding	2022-4-9

Introduction

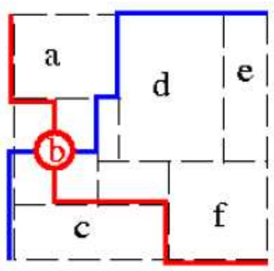
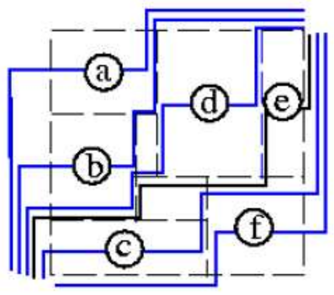
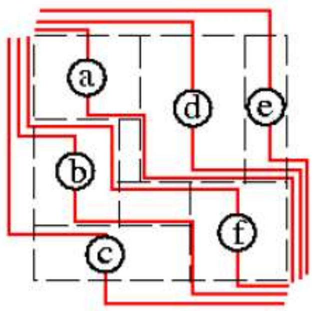
This is a simplified implementation *FAST-SP: a fast algorithm for block placement based on sequence pair*. I did not deal with the *pre-place constraint*, *range constraint*, *boundary constraint* described in the paper.

However, my code can generate a floorplan within a given chip boundary *(W, H)*. For more detailed spec information, please refer to README.

Sequence Pair

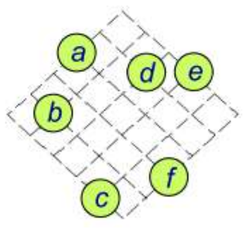
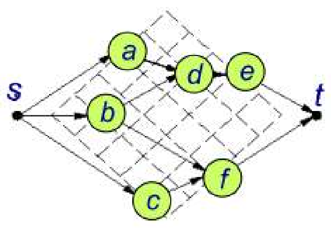
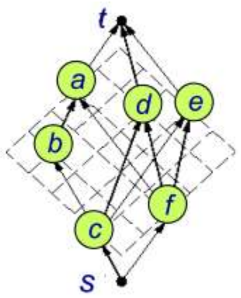
Sequence Pair (SP) is a floorplan representation by a pair of module-name sequences: positive locus, and negative locus. It does not gaurantee the floorplan to be compacted. Yet, it has a P*-admissible solution space.

For detailed information, please refer to the references.

		
Loci of module b *- from Handout unit4 -*	Positive loci : ***abdecf*** *- from Handout unit4 -*	Negative lock : ***cbfade*** *- from Handout unit4 -*

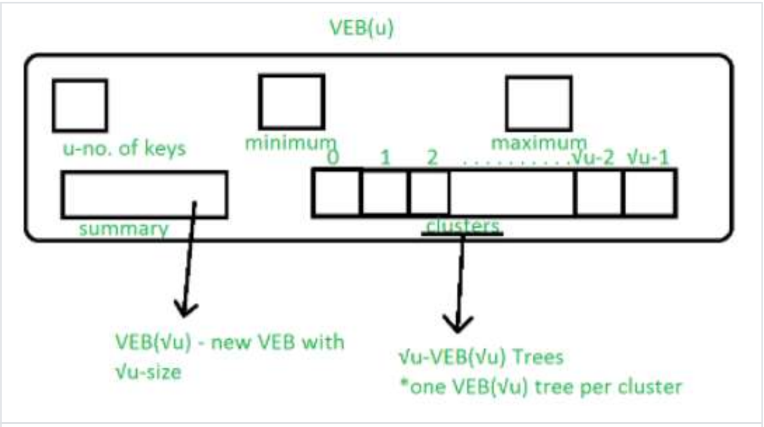
In the paper *Rectangle-packing-based module placement*, the packing is obtained by longest path algorithm on two vertex weighted directed acyclic graph: a horizontal constraint graph, and a vertial constraint graph.

This method is in $O(n^2)$.

		
Packing for SP *- from Handout unit4 -*	Horizontal constraint graph *- from Handout unit4 -*	Vertial constraint graph *- from Handout unit4 -*

In the paper *FAST-SP: a fast algorithm for block placement based on sequence pair*, the packing is computed by the longest common subsequence of the two sequences.

This method is in $O(n \lg \lg n)$ with the help of the special data structure: Van Emde Boas Tree.


VEB(u)
u-no. of keys
minimum
maximum
0 1 2 .... √u-2 √u-1
summary
clusters
VEB(√u) - new VEB with √u-size
√u-VEB(√u) Trees
*one VEB(√u) tree per cluster

Structure of Van Emde Boas Tree
- *from GeeksforGeeks* -

I applied the longest common subsequence with $O(n \lg \lg n)$-time complexity in my implementation.

Data Structures

- In `./src/module.h`
  - `class Terminal` : Information of terminals, including their name and coordinates.
  - `class Block` : A derived class of `class Terminal` which includes addtional information of a block : rotation, width, and height.
  - `class Net` : Information of nets, including a list of terminals connected and another list of blocks connected.
- In `./src/veb.h`
  - `class VanEmdeBoas` : Van Emde Boas Tree structure. Note that the code is directly copied from GeeksforGeeks Van Emde Boas Tree | Set 4 | Deletion. Because it is a special data structure, I suppose it is okay to copy it.
- In `./src/sp.h`
  - `class SequencePair` : Using Sequence Pair as the floorplan representation and applying Simulated Annealing for optimization. Below are some important functions:
    - `SequencePair.Solve()` : Solve the fixed-outline floorplanning problem by first calling `SequencePair.RandomInitialize()` to generate an initial floorplan and then applying Simulated Annealing for further optimization.
    - `SequencePair.RandomInitialize()` : A random initialization mechanism that generates 100 random floorplans by shuffling the sequence pair elements. The floorplan with the minimum area size will be the final initial floorplan.
    - `SequencePair.EvaluateSequence()` : An algorithm from *FAST-SP: a fast algorithm for block placement based on sequence pair* that uses the longest common subsequence algorithm and Van Emde Boas Tree structure to compute the coordinates for every blocks.

Simulated Annealing

Simulated Annealing is implemented in `SequencePair.Solve()` based on the pseudo code in Handout unit4 pg 9.

Here is the pseudo code of my modified version.

```
S = random_initial_solution
T = 99999999.0
R = 0.999
NUM_STEPS = 10
strike_cnt = 0
while (true):
    if (T < 1)
        if (has_found_legal_solution)
            break
        else
            T = 999.0
    for (1 <= i <= NUM_STEPS)
        S' = random_neighbor(S)
        delta = cost(S') - cost(S)
        if (has_found_legal_solution == false)
            if (is_legal(S') == true)
                T = 99999.0
        if (delta <= 0)
            S = S'
        else
            S = S' with probability e^(-delta/T)
    if (is_change(S) == false)
        strike_cnt++
    if (strike_cnt > 5000)
        break
    T = R * T
return S
```

- The paper *FAST-SP: a fast algorithm for block placement based on sequence pair* mentioned that they used Simulated Annealing with a very large number of tempaeratures and a small number of moves within each temparature. Therfore, I set the initial temparature to `99999999.0`, the decading variable to `0.999`, and the number of moves per temparature to `10`.
- If the solver has not found any legal solution yet and the temparature has drop under the frozen condition, I reset the temparature to `999.0` to let the optimization keep going (line 7 ~ 10).
- In case the optimization end too early, I reset the temparature to `99999.0` once the solver find its first legal solution (line 15 ~ 17).
- `strike_cnt` can terminate the optimization process earlier if no further optimization is made, which enhance the timing performance.

Coordinates Computation

Coordinates Computation is implemented in `SequencePair.EvaluateSequence()` based on the paper *FAST-SP: a fast algorithm for block placement based on sequence pair*. The $O(n \lg \lg n)$-time method utilize the longest common subsequence algorithm and Van Emde Boas Tree structure.

Here is the pseudo code from the paper.

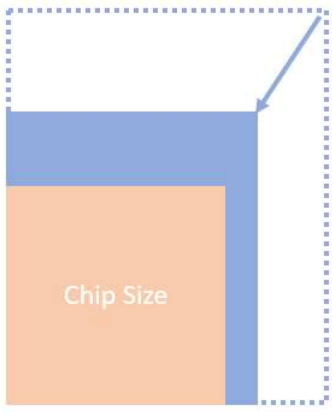
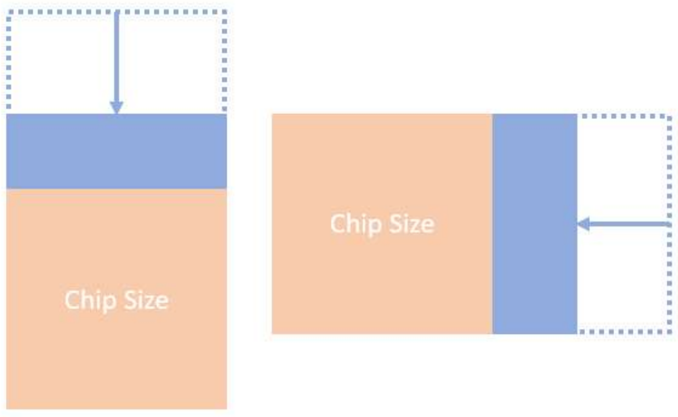
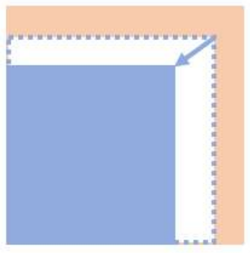
```
Algorithm Eval_Seq(X, Y)
    Initialize_Match_Array MATCH
    Initialize H, insert the initial index 0
    Initialize BUCKL with BUCKL[0]
    for (1 <= i <= n)
        b = X[i]
        p = MATCH[b].y
        insert p to H and BUCKL
        POS[p] = BUCKL[predecessor(p)]
        BUCKL[p] = POS[p] + w(b)
        Discard the successors of p from H and BUCKL whose value <= BUCKL[p]
    return BUCKL[max_index]
```

- `H` is a Van Emde Boas Tree structure implemented in `./src/veb.h`.

Findings & Optimization

Except for the optimization mechnisms mentioned above, there are some other methods I applied in my code.

- Only accept legal solution after one legal solution has been found.
- In the paper *FAST-SP: a fast algorithm for block placement based on sequence pair*, it mentioned that they decreased *W* and *H* when a feasible wequence pair is met. I implemented this by only accepting the downhill moves after any legal solutions has been found.
- For *ami49*, the Simulated Annealing is hard to find a legal solution within the given chip boundary. I notice that it iss because the original cost function only consider the total area. Therefore, the optimization tends to make the floorplan *square*, whereas the boundary constraint for *ami49* is rather *rectanglular*. Hence I modified the cost function into three stages: (1) When both width and height exceed the chip boundary, the cost is the area size. (2) When width exceeds the boundary constraint and height does not, the cost is the width, and vice versa. (3) When both width and height are inside the chip boundary, again, the cost is the area size. In this way, my implementation can find a legal solution for *ami49*.

		
cost = area	*cost = width or height*	*cost = area*

Here are my findings:

- Sequence Pair with the longest common subsequence algorithm and Van Emde Boas Tree structure is really fast. For example, it can generate and compute nearly 145,000 floorplans within 24.41 seconds for *ami49*.
- The solution quality and running time of my results outperform those of the results from others (my friend or internet).

Experiment

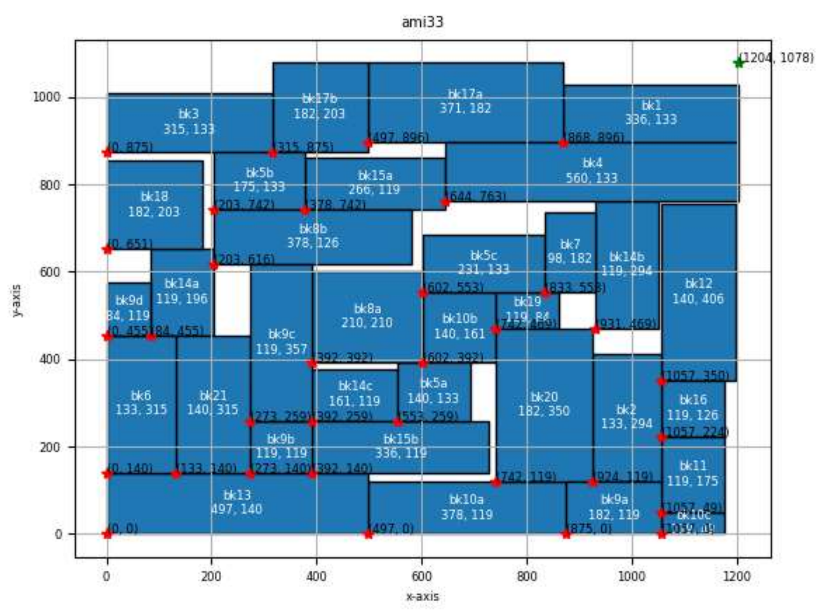
- All testcase can find a legal solution.
- My optimization only considers area and does not condsider wirelength. That is, the alpha value does not affect my result.
- The solutions of the same testcase may vary from each execution because of the random operations in Simulated Annealing.
- For testcase *ami49*, it may take much longer time to find a legal solution when the random operations are with bad luck.

Results

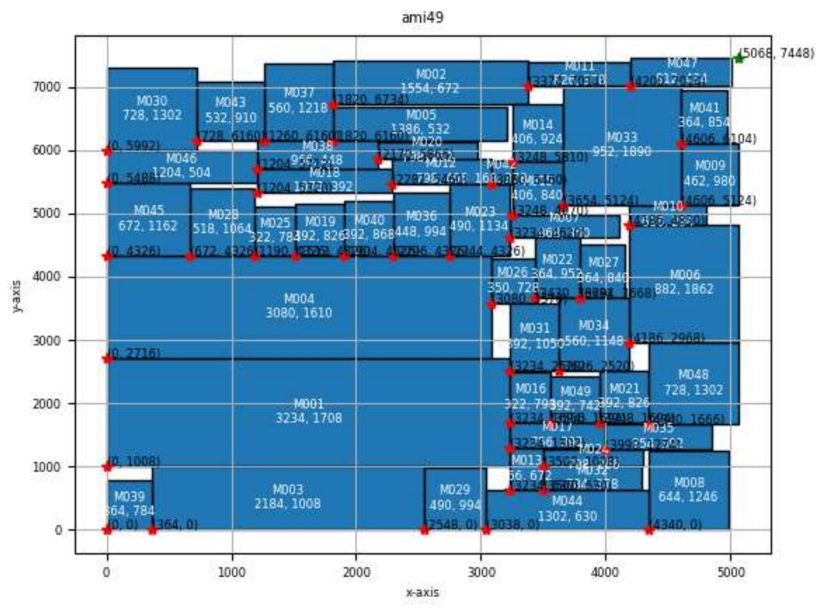
testcase	outline width	outline height	chip width	chip height	chip area	total wirelength	cost $\alpha = 0.5$	runtime (s)
ami33	1326	1205	1204	1078	1297912	124551.5	711231.75	12.54
ami49	5336	7673	5068	7448	37746464	1892576.0	19819520.0	24.41
apte	11894	6314	9478	5490	52034220	997334.0	26515776.5	0.53
hp	5412	3704	3892	2520	9807840	314478.0	5061159.5	0.62
xerox	6937	5379	5264	3885	20450640	686979.0	10568810.0	0.57

Floorplan visualization

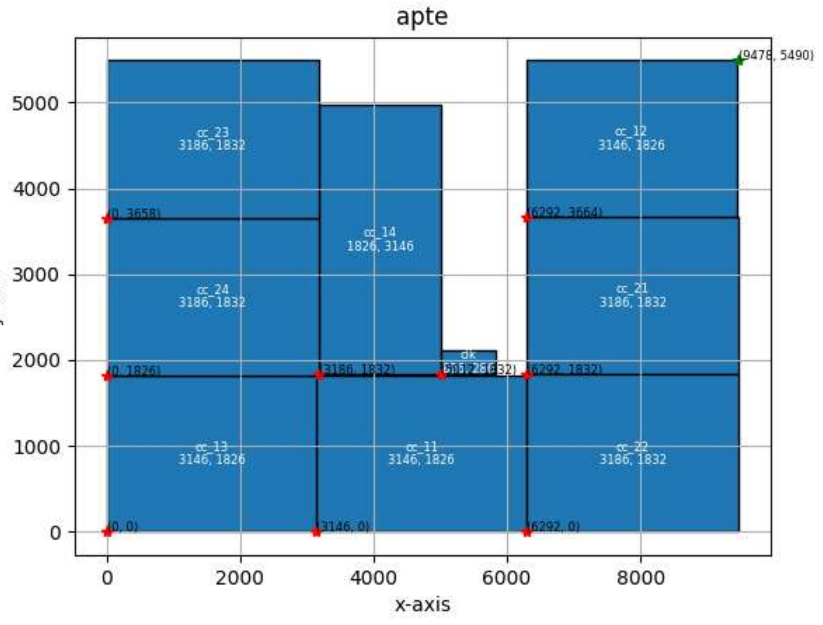
I wrote a python program `./src/plot.py` to visualize the generated floorplan.



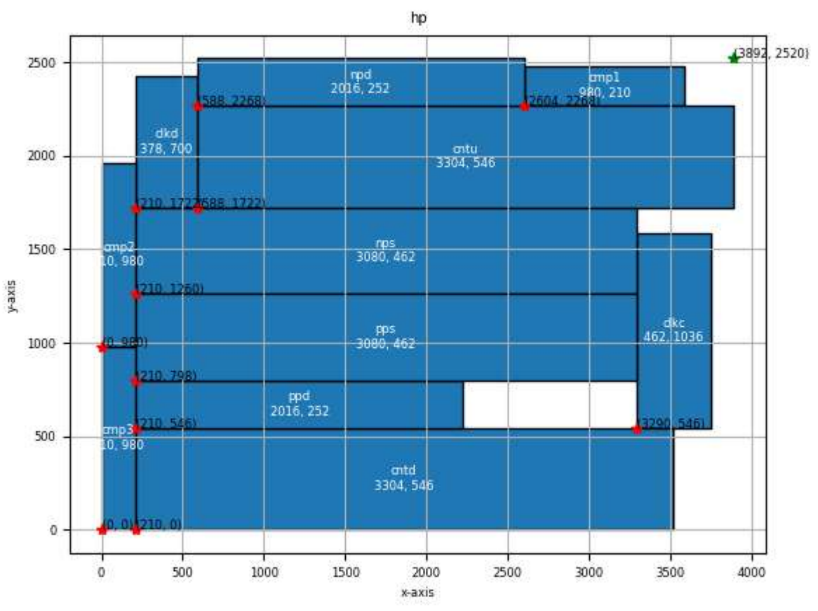
ami33


ami49
y-axis
x-axis
M030
728, 1302
M043
532, 910
M037
560, 1218
M002
1554, 672
M011
M047
(5068, 7448)
M005
1386, 532
M014
406, 924
M033
952, 1890
M041
364, 854
M046
1204, 504
M038
956, 448
M020
M012
M042
M009
462, 980
M018
406, 840
M045
672, 1162
M028
518, 1064
M025
322, 784
M019
392, 826
M040
392, 868
M036
448, 994
M023
490, 1134
M010
M007
M004
3080, 1610
M026
350, 728
M022
364, 952
M027
364, 840
M006
882, 1862
M031
392, 1050
M034
560, 1148
M001
3234, 1708
M016
322, 798
M049
392, 742
M021
392, 826
M048
728, 1302
M017
M035
M013
M024
M032
M039
364, 784
M003
2184, 1008
M029
490, 994
M044
1302, 630
M008
644, 1246
(0, 5992)
(0, 5488)
(0, 4326)
(0, 2716)
(0, 1008)
(0, 0)
(364, 0)
(2548, 0)
(3038, 0)
(4340, 0)
(728, 6160)
(1260, 6160)
(1820, 6160)
(1820, 6734)
(672, 4326)
(3248, 5810)
(3654, 5124)
(4606, 6104)
(4606, 5124)
(4186, 2968)
(4340, 1666)
(3234, 2520)

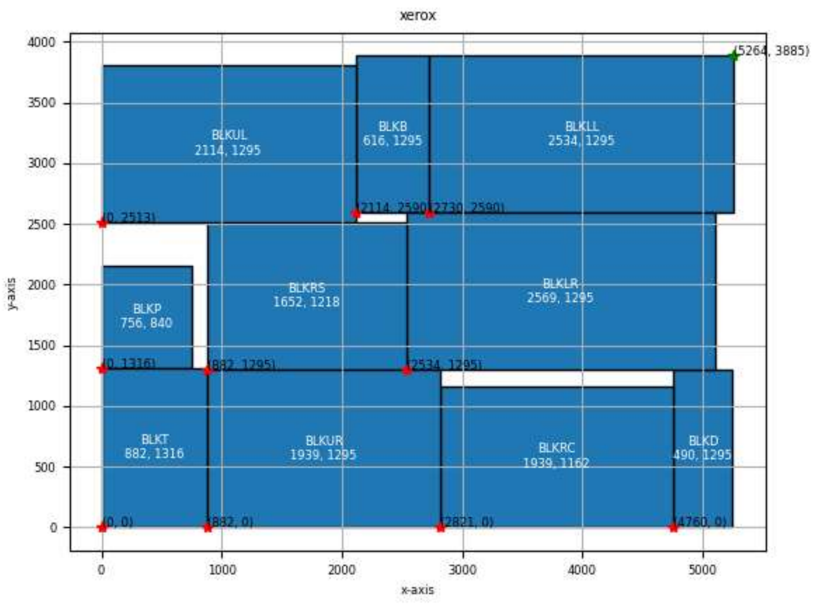
ami49


apte
y-axis
x-axis
cc_23
3186, 1832
cc_14
1826, 3146
cc_12
3146, 1826
cc_24
3186, 1832
cc_21
3186, 1832
clk
cc_13
3146, 1826
cc_11
3146, 1826
cc_22
3186, 1832
(9478, 5490)
(0, 3658)
(6292, 3664)
(0, 1826)
(3186, 1832)
(6292, 1832)
(0, 0)
(3146, 0)
(6292, 0)

apte



hp



xerox

Reference

- Sequence Pair
  - Handout unit4 pg36 ~ pg40
  - *Rectangle-packing-based module placement*
  - *FAST-SP: a fast algorithm for block placement based on sequence pair*
- Van Emde Boas Tree - *from GeeksforGees* -
  - Van Emde Boas Tree | Set 1 | Basics and Construction
  - Van Emde Boas Tree | Set 2 | Insertion, Find, Minimum and Maximum Queries
  - Van Emde Boas Tree – Set 3 | Successor and Predecessor
  - Van Emde Boas Tree | Set 4 | Deletion