B07901026 吳禎軒 PA2 Report

$Data Structure and how I designed the code:

1. Object:

(a)Terminal: store x and y coordinate for a terminal

(b)Net:

#x_min, _y_min, _x_max, _y_max- possible candidates involved in calculating HPWL, and their defaults are the coordinates of terminals in that Net

#Blocklist- integer vector storing the indexes of involved blocks

(c)Block:

#basic information- width, height, location

#position in B* tree- current parent, current left and right child

#feasible and best position in B* tree- record the previous information for the conditions, such as rejection and best case retrieval

#role=x- if x==-1, the block is the root; if x==0, the block is a left child; if x==-1, the block is a right child

2. Data structure of class B* tree:

_blockArray, _termArray, _netArray- store the objects mentioned above

#mapping from ID to name- used for output

_maxX, _maxY- current boundary

_root- current root of B* tree

_hContour(map<int a, int b>)- record the horizontal contour; the height of the right side of the line x=a is b

#feasible and best condition- record the previous information

3. Initialization: This part will be discussed in next section.

4. Update y coordinate of a block by horizontal contour

There are 3 cases(define lx/rx the left/right side of the new block)-

(a)both lx and rx are on the right side of _maxX-

The new block is adjacent to the rightmost block, and its y coordinate is its height.

(b)lx is on the left side of _maxX, rx is on the right side of _maxX-

y coordinate of new block= its height+ the maximum height of a candidate set including the rightmost block on the left side of lx and all blocks between lx and rx

(c) both lx and rx are on the left side of _maxX-

It's similar to case (b), but updating _hContour is more complicated.

5. Tree Traversal: preorder traversal

6. Three operations for modifying tree structure:
(a)rotation: swap the height and width of the block
(b)swap: Forbid swapping two nodes whose relationship are parent-child
(c)deletion and insertion: Delete a node and let the leftmost leaf node be its successor. Then, insert it to the left or right child of the target node, depending on the number of target node's children.
7. Simulated Annealing:
Use rand() to select one operation, and I use pseudo random number here so that debug is easier. Accept the new tree iff the _deltacost is negative because uphills create a worse solution and take longer time. Run SA until a legal solution is found. A_{norm} and W_{norm} for cost function are the average of all samples, including rejected ones.
#initial temperature T1 = Δ/lnP, where Δ = 0.01 and P = 0.99
#freezing factor is 0.85
#frozen temperature is 0.00008
#reject rate threshold is 0.95
#Inner while loop move limit is 2*problem size

$My finding

1. Initialization is important

I notice that initialization affects the final area and the running time significantly. The following are different methods I've tried:
(a)Build a complete binary tree following the order in input file
This method is suitable for smaller cases, and I guess the reason is that the initialization tree structure is already close to the final solution.
(b)Flip the block if its width is smaller than its height
Combined with (a), the performance is poor.
(c)Sort the width of all blocks and insert them layer by layer
I think combining this method with (b) should be the most efficient one, but it still fails to generate a legal solution. There are some mistakes in my program, and I have inadequate time to fix them.
(d)Manual perturbation in initial tree structure
For ami33 and ami49, I cannot find a legal solution using (a)~(c). I pack blocks with similar sizes together and send a better tree structure to SA to find a better solution. This method is not a good idea for hidden cases or general testing. Besides, it has a problem: For different alpha values, the final solutions are almost identical. I guess the reason is that the initial tree structure is not flexible enough so that the operations in SA never reaches the feasible condition. A new set of parameters for SA may be helpful.

2.What should we do when SA fails to find a solution to a larger case?

I solve the problem by generating a better initialization. Before that, I have tried different methods:

(a)Tree traversal: Use BFS instead of preorder traversal. It is not helpful for larger cases and has terrible performance for smaller cases.

(b)Cost function: I tried setting different calculation methods for Anorm and Wnorm, such as the average of accepted samples and the initial area/ wirelength, but the problem still cannot be solved.

(c)Operation selection: Use a more random generator(srand(time(NULL))) or increase the probability of insertion, which is fruitless.

(d)Parameters for SA: I've tried fast SA or other values for temperature and move limits. It doesn't work for larger cases.

These methods should work theoretically, but I am not able to spend longer time figuring out how to use them properly and effectively.

3. Debug and Validation

I've found that it's necessary to print out the horizontal contour, the tree structure, and other detailed information to check the correctness of the program. Many checking functions are enclosed in module.cpp . Also, I use GUI (plot.m) to show the floorplanning results, which makes it easier to check the horizontal contour.

4. Challenge: Mediocre area and wirelength

My program runs quite fast compared with the students last year, but the solution quality is not high. Although I've spent 10-12 hours every day this week writing this program, it is worthwhile to invest more time to improve the performance.