B07901026 吳禎軒 PA1 Report

$Data Structure and how I designed the code:

1. The class for Cell, Net, Partitioner are based on TA's codes. The algorithm is based on FM algorithm, and I make some modification.
2. Initialization: This part will be discussed in next section.
3. Bucketlist: map<int,Node*>

To move a node, delete it from its original gain and insert it to its new gain.
For a node N1 whose gain is G1 ,
Deletion has four cases: N1 is the only cell whose gain is G1, N1 is the head but not the only member of the bucket, N1 is the tail but not the only member of the bucket, N1 has its previous and next member.
Insertion has two cases: G1 does not exist in the bucket list, G1 exists in the bucket list

4. _maxGainCell : two Node* pointing to the node whose gain is max for each part

maxgain: an array whose size equals to 2 saves the value of max gain for each part

5. Keep the partition balanced: In the process of initialization and each step in one iteration, check whether the partition is balanced. Besides, choose the more balanced solution if the two cells in different parts have same gain for each step of update. Selecting accumulated gain is similar - the more balanced solution will be selected if there are same accumulated gain in two conditions.
6. Update maxgain: This part will be discussed in next section.
7. Reset for each iteration: Empty the bucket list, unlock all cells, reset all cells' gain, find the partition for the best move

$My finding

1. Initialization is important

I notice that initialization affects the final cutsize and the running time significantly. The following are different methods I've tried:

(a)Divide according to the number of pinnumber of each net

This method is suitable for larger cases or cases whose nets have unbalanced sizes, such as input_0, input_5, avoiding cutting larger nets. However, it takes a little more time to sort net size.

(b)The first half of cells are in the same side, the others are in the other side

Naively attempting to reduce the probability of cutting a net may unexpectedly have better performance for some cases, such as input_4.

(c)Divide randomly (multiple of 2)
This method has terrible performance for most cases, especially for input_0. However, a more complex combination may be suitable for smaller cases, such as input_1 and input_2, which results in a much smaller cutsize. I have tried this method for some cases, but it is not a good idea for hidden cases or general testing.
(d)In each step of dividing, put the cell to the side where there are more connections at that moment
This is TA's suggestion. I think this is an effective approach that is most likely to obtain smaller cutsize, but my poor programming skills causes the program to take too much to run.

2. How to update maxgain efficiently for each update?

There are two cases for updating maxgain according to TA's suggestion:
(a) the gain of one cell exceeds the previous maxgain -> that cell becomes _maxGainCell
(b)no gain of any cell exceeds the previous maxgain -> search the bucket list by starting from previous maxgain and decreasing the pointer gradually
I encountered a bug here because I checked (a) every time when I added 1 to a cell's gain: the change of a cell's gain may be [+1,-1,-1], so selection between (a) and (b) will be misjudged. Thus, I solved the problem by checking whether the temporary _maxGainCell are qualified upon subtracting 1 from its gain.

3. Debug and Validation

I've found that it's necessary to print out the gain, part, and other detailed information to check the correctness of the program. Many checking functions are enclosed in partitioner.cpp . Also, I wrote one simple file that can check the correctness by examining only the input file and the output file.
Checker - http://codepad.org/OrGtk9AP

4. Challenge: high running time

My program has much higher running time compared with my classmates. I guess the reason may be that each reset clears the bucket list and requires rebuilding for each step in one iteration, but I have inadequate time for improvement.