# Programming Assignment 1 b05901025 王鈺能

### A. Algorithm Flow

Following FM heuristic discussed in class, my algorithm flow is described below:

// initialPartition() divides cells into to group based on randomness   
// initialGain() calculates cells' gain by identifying critical nets // and increase/decrease gain of cells on the net   
// getMaGainCell() returns a free cell with maximum gain   
// undateGain() updates cells'gain by checking nets' from-side and to // side for nets in the cell's terminal   
// updateSum() updates current sum of gain, maximum gain, and which // step to terminate   
// (corresponding to maximum gain)   
// After iterating through every cell, bestMove() moves cell to .// corresponding group   
// Iteration end when maxGain < 0(but in fact, will never occur) or // time limit exceed   
initialPartition();   
while (1)   
{   
 initialGain();   
 gainSum = 0;   
 maxGainSum = -1;   
 terminateStep = -1;   
 for(int i = 0; i < cells.size(); i++)   
 {   
 lockedCell = getMaxGainCell();   
 lockedCells[i] = lockedCell;   
 lockedGroup[i] = ( lockedCell->belong == &groups[1] );   
 updateGain( lockedCell );   
 updateSum( lockedCell, gainSum, maxGainSum, terminateStep, i );   
 }   
 if ( maxGainSum > 0 || ( maxGainSum == 0 && !TLE() ) ) bestMoves( lockedCells, lockedGroup, terminateStep );   
 else break;   
}

### B. Data Structure

I model this problem using 4 structure

##### a. Cell

class Cell  
{  
friend class Group;  
friend class Circuit;  
private:  
 Nets terminals; // Nets connect to this cell  
 // Nets = std::vector<Net\*>  
 CellListIt gainPos; // Pointer point to this cell's position in the bucket list  
 Group\* belong; // Which group(bucket) this cell belongs to  
 int gain; // Current gain for moving this cell  
 char\* name; // Cell's name  
public:  
 Cell () { gain = 0; }  
 ~Cell () {}  
  
 // updating function ...  
 // access function ...  
 // print function ...  
};

##### b. Net

class Net  
{  
private:  
 Cells terminals; // Cells connect to this net  
 // Cells = std::vector<Cell\*>  
 int gCount[2]; // Number of Cells in each group  
 // Usage: for update gain  
 char\* name; // Net's name  
public:  
 Net () {}  
 ~Net () {}  
  
 // updating function ...  
 // access function ...  
 // print function ...  
};

##### C. Group

class Group  
{  
private:  
 CellList \*GainArray; // Lists to store cells based on their gain  
 // CelList = std::list<Cell\*>  
 // CellListIt = CellList::iterator  
 int Pmax, max, gsize; // Pmax: max terminal size of cells  
 // max: current max gain in list  
 // gsize: number of cells in this group  
public:  
 Group () {}  
 ~Group() {}  
  
 // updating function ...  
 // access function ...  
 // print function ...  
};

##### D. Circuit

class Circuit  
{  
private:  
 Cells cells; // All cells in the circuit  
 Nets nets; // All nets in the circuit  
 Group\* groups; // 2 partition group  
 double ratio; // balance constraint  
  
public:  
 Circuit () {}  
 ~Circuit () {}  
   
 // FM function  
 // updating function ...  
 // access function ...  
 // print function ...  
};

### C Discussion

##### a. Termination Condition

Usually, FM heuristic will terminate when maximum gain equals to. However, I found out that max gain might be positive after some iteration after 0 gain. Hence, even with gain = 0, I won't terminate my process, just trying to find more possibly better partition. Considering timing, I also add an termination condition based on log(circuit size).

##### b. Random Initialize

Initial condition play a vital role in FM heuristic. While first partition, I use rand()%2 to determine every cell's group, and try many different seed to get a better result.

##### c. Random Max Gain Cell

After executing many iterations, the partition will converge to some local minimum.To jump out of it and go to a better minimum partition, I add some randomness in getting max cell in bucket list. Originally, i take the first element of the list as max cell. Here I return the rand()%20 cell in the list (if not end). Because 20 is constant, The "get max cell" is still performed in constant time, maintaining the property of the bucket list.