PA3 Report

B09901066 電機三 謝承修

在這份作業中，我原本是想用ePlace來實作，後來研究了一下發現好複雜，所以我後來改成用隨機的，然後再用一些heuristic的想法去做，效果竟然也還行，但也花了我不少時間。

主要可以分為以下幾個部分：

1. 先將連結較為緊密的module cluster起來。想法很簡單，如果連結較緊密的modules擺在一起可以減少HPWL。而我這邊的實作是先將nets根據pin的數量由大排到小，並用一維向量"_order"去紀錄這些nets中的module，若重複則跳過。會用一維向量的原因是我只需要紀錄哪些要先放就好了，等下的步驟會看到。

```
////////// Cluster Start //////////
set<int> _check;
vector<int> _order;  // the order of placeing the modules
vector<Net> nets = _placement.getNets();

// sort the nets by the number of pins
sort(nets.begin(), nets.end(), [](Net& a, Net& b) {
    return a.numPins() > b.numPins();
});

for (int i = 0; i < nets.size(); i++) {
    int numPins = nets[i].numPins();
    for (int j = 0; j < numPins; j++) {
        // go through each module in the net
        int moduleId = nets[i].pin(j).moduleId();
        // check if the id is already in _check set
        if (_check.find(moduleId) == _check.end()) {
            _check.insert(moduleId);
            _order.push_back(moduleId);
        }
    }
}
////////// Cluster End //////////
```

```
std::uniform_real_distribution<double> distribution(0.0, 1.0);
std::default_random_engine generator(param.best_seed);
int binWidth = chip_width / param.best_num_section;
int binHeight = chip_height / param.best_num_section;
int binSize = num_blocks / (param.best_num_section * param.best_num_section);
vector<double> sortX;
vector<double> sortY;
for (int i = 0; i < param.best_num_section; ++i) {
    int idx_x = (i + param.best_num_section / 2) % param.best_num_section;
    for (int j = 0; j < param.best_num_section; ++j) {
        for (int k = 0; k < binSize; ++k) {
            double randomX = left_bound + binWidth * idx_x + binWidth * distribution(generator);
            double randomY = bottom_bound + binHeight * j + binHeight * distribution(generator);
            sortX.push_back(randomX);
            sortY.push_back(randomY);
        }
    }
}
// assign the coordinates to each module
for (int i = 0; i < num_blocks; i++) {
    x[_order[i] * 2] = sortX[i];
    x[_order[i] * 2 + 1] = sortY[i];
}

NumericalOptimizer no(ef);
no.setX(x);  // set initial solution

for (int i = 0; i < num_blocks; i++)
    _placement.module(i).setPosition(no.x(i * 2), no.x(i * 2 + 1));
```

2. 接著將先將整個chip分割成許多bin，然後每個bin中再產生binSize個座標點。會選擇這樣做的原因是可以確保placement的結果是分散的，並且容易將連結性強的modules擺在一起。這邊的擺法我有實驗三種。第一種是從左下排到右上，第二種是從中間開始排，排到最右邊之後再從最左邊排到中間，第三種是從正中心開始螺旋狀往外排(code如右下)。我原本以為是第三種會達到比較好的結果，因為很直覺連結度最高的要盡量擺在中間，結果我試過很多組參數，發現不管怎麼擺，都是第二種比較好，所以我最後是implement第二種方法。

3. 基本上到這邊就差不多了，已經可以跑出不錯的成績。但我秉持著實驗的精神，還是寫迴圈測了很多不同的參數。主要就是寫在ParamPlacement.h中的best_seed及best_num_section(一邊要分割成幾段)。然後我這邊花了超多時間在測參數的。因為我發現如果把detail placement放在迴圈裡面，最後單獨測seed的話答案會不一樣，超扯，推斷應該是detail placement裡面有用到隨機性相關的東西。然後為什麼要測到detail placement呢？因為我原本只測global placement，後來發現global placement得出的HPWL往往跟最後detail placement後的結果沒有正相關，有可能得到一組很屌的Global placement擺法，但是最後detail placement得到的HPWL有沒很好。我猜是detail placement太強了，會對前面得到的placement有大幅的優化。所以實驗必須也要考慮detail placement的部分。

```
// generate random coordinates for each bin in a spiral pattern
int centerX = num_section / 2;  // center X coordinate
int centerY = num_section / 2;  // center Y coordinate
int currentX = centerX;
int currentY = centerY;
int direction = 0;  // 0: right, 1: up, 2: left, 3: down
int steps = 1;      // number of steps in the current direction
int stepCount = 0;  // count of steps taken in the current direction

for (int i = 0; i < num_section * num_section; ++i) {
    for (int k = 0; k < binSize; ++k) {
        double randomX = left_bound + binWidth * currentX + binWidth * distribution(generator);
        double randomY = bottom_bound + binHeight * currentY + binHeight * distribution(generator);
        sortX.push_back(randomX);
        sortY.push_back(randomY);
    }

    // move to the next coordinate in the spiral pattern
    switch (direction) {
        case 0:  // right
            currentX++;
            break;
        case 1:  // up
            currentY--;
            break;
        case 2:  // left
            currentX--;
            break;
        case 3:  // down
            currentY++;
            break;
    }

    stepCount++;
    if (stepCount == steps) {
        stepCount = 0;
        direction = (direction + 1) % 4;        // change direction
        if (direction == 0 || direction == 2) {  // increase steps after moving right or left
            steps++;
        }
    }
}
```

在這次作業中，我並沒有用到什麼特殊的資料結構，主要就是在placement.h, Globalplacement.h, Global placement.cpp中加入一些函數來輔助計算module的位置。

這次學習比較多的地方是怎麼調參數，感覺好像是我之前在修ML在做的事。然後前述說到如果在main.cpp中寫迴圈跑參數，會有最後不可reproduce的情況，所以我直接寫了一個shell script去call程式，然後把結果額外記錄在別的txt檔案，最後再用python去找哪組參數比較好。然後我有發現一件蠻重要的是，就是不同的testcase中bin切成不同大小會影響HPWL蠻多的，比seed還重要，例如在ibm01的測資中，如果每邊都切成55份，HPWL會明顯變好，推測是因為和一個net平均連到多少module以及chip size有高度相關。如果助教有興趣的話可以看我檔案裡面的ibm0X.txt檔。

我原本以為我用隨機的話可以省不少時間弄別科的東西，後來發現測這些也好麻煩，但也算是有學到一些東西啦。如果暑假有時間的話應該會用ePlace重寫一次，畢竟聽學長說這個蠻屌的。