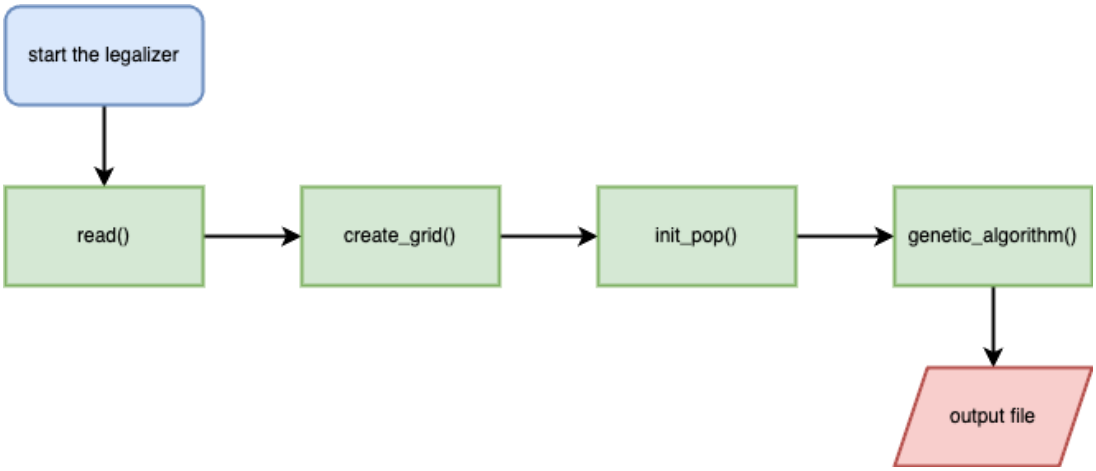
Final Project Report

112062554 劉廷宜

Overall Design Flow



在這個project中，我先greedy的幫每個instance找到距離相對近的resource擺放，再利用基因演算法來優化solution
先來大致說明每個大步驟，之後會在Implementation再詳細說明

`read()`：讀入architecture/instance/netlist這三個input檔案並建構所需要的資料結構
`create_grid()`：將讀入的CLB/RAM/DSP resource用座標排序後生成矩陣以便之後instance可以在最初legalization的時候找到距離較近的resource擺放
`init_pop()`：初始化基因演算法所需要的基本population
`genetic_algorithm()`：利用crossover/mutation等操作來優化solution

Implementation

- **Data Structure**

```
enum blockType {
    IO,
    CLB,
    RAM,
    DSP
};
```

為了之後可以方便得到每個resource/instance的型別數字，所以我使用了enum來表示四種type的數字

```
struct Block {
    std::string name;
    int type;
    double center_x, center_y;
    int id;
    Block(std::string n, int t, double x, double y) : name(n),
```

```
type(t), center_x(x), center_y(y) {}
};
```

Block用來記錄resource/instance的訊息，其中紀錄了它們的名字、型別、座標等

```
struct Net {
    std::string name;
    std::vector<Block*> inst_vec;
    Net(std::string name) : name(name) {}
};
```

Net用來記錄每個net的名字以及所包含的instance block

```
struct gene {
    double fitness;
    std::vector<int> resource_permu[4];
};
```

gene是基因演算法中的一組基因，裡面包含他們的fitness value以及CLB/RAM/DSP各個resource的permutation狀態
（可以想像instance的index都是不變的，而是resource會打亂去assign給instance因而得到不同的結果）

- **read()**

```
std::vector<Block*> resource[4];
std::vector<Block*> inst[4];
std::vector<Net*> net_vec;
```

主要讀入三個檔案來建構上方的資料
resource/instance相同，都是利用vector來記錄Block
再藉由剛剛說明的enum就可以得到特定型別的資料
（例如要得到CLB的resource資料就是resource[blockType::CLB]）

```
std::unordered_map<std::string, Block*> nameToInst;
```

在讀入第二個檔案時會順便紀錄每個instance的名字映射到的instance block
以便讀入第三個檔案時建構每個net包含的instance block

- **create_grid()**

```
std::vector<std::vector<Block*>> resource_grid[4];
```

這個function是將每個型別的resource藉由座標的排序來建構一個二維陣列
以下就以CLB來說明是怎麼做的：

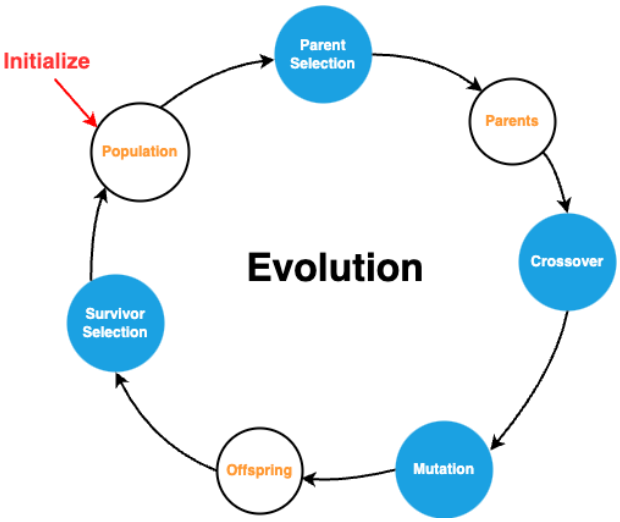
```
tmp[blockType::CLB] = resource[blockType::CLB];
stable_sort(tmp[blockType::CLB].begin(), tmp[blockType::CLB].end(), []
(const Block* lhs, const Block* rhs){
    return lhs->center_y < rhs->center_y;
});
```

先將resource依y座標由小到大排序

```
// initialize h = min y-coordinate of CLB
for (auto &it : tmp[blockType::CLB]) {
    if (it->center_y != h) {
        // sort the vector by x-coordinate
        resource_grid[blockType::CLB].emplace_back(v);
        v.clear();
        v.emplace_back(it);
        h = it->center_y;
    } else {
        v.emplace_back(it);
    }
}
```

上面這段程式碼主要就是將同樣高度的block放入一個vector中
直到遇到不同的高度就將那個vector中的block依x座標由小到大排序並放入resource_grid中

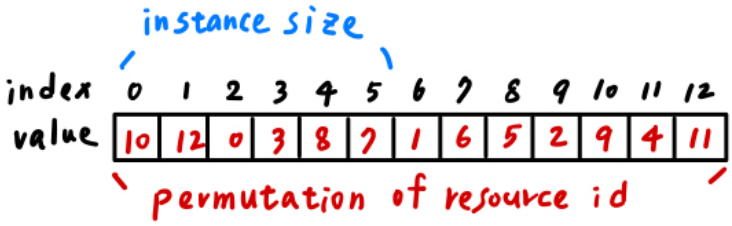
Overview of Genetic Algorithm



基因演算法的流程基本如上圖
首先需要初始化population，而population是由許多的gene組成
基因可以想像成一個一維陣列，看題目的需求定義representation
之後的流程都是一樣的，直到做完一定回數或是時間限制到了而停止

流程如下：
(1) 從population中挑選兩個gene作為parents
(2) 對parents做crossover後產生兩個子代
(3) 再對兩個子代各自做mutation
(4) 重複(1)-(3)直到產生 (population size)/2 個子代
(5) 將原先的population加上後來的子代做處理形成新的population
(6) 重複(4)

Representation



在 `read()` 我會先將resource/instance依照型別分組並編號
可以看上圖，index可以想為instance的編號，而在index位置儲存的value為resource的編號
而index = i儲存value = j的意思就是第i個instance會放置到第j個resource中
所以經由不同的resource id的permutation就可以得到不同的legalization結果

- **init_pop()**

```
const int POP_SIZE = 100;
```

在這個function中，要產生初始的POP_SIZE個基因到pool中
首先利用`create_grid()`建構出的二維陣列，計算出instance距離哪個resource最近

```
int dist = 1;
int dx[8] = {0, 1, 0, -1, 1, 1, -1, -1};
int dy[8] = {1, 0, -1, 0, 1, -1, -1, 1};
while (1) {
    for (size_t i = 0; i < 8; ++i) {
        if (exceed the range) continue;
        if (resource_grid[type][r + dist * dy[i]][c + dist * dx[i]] is
not occupied) {
            put the resource id to the resource_permu[type][idx] and
return
        }
    }
    dist++;
}
return 0;
```

如果那個resource已經被別的instance佔用的話，就以那個resource為中心向外擴散尋找還沒被佔用的resource

```
for (auto &g : pool) fitness(g);
std::sort(pool.begin(), pool.end(), [](const gene& lhs, const gene&
rhs){
    return lhs.fitness < rhs.fitness;
});
```

最後會計算所有gene的fitness（就是那個gene中legalization結果得到的HPWL）並由小到大排序

- **genetic_algorithm()**

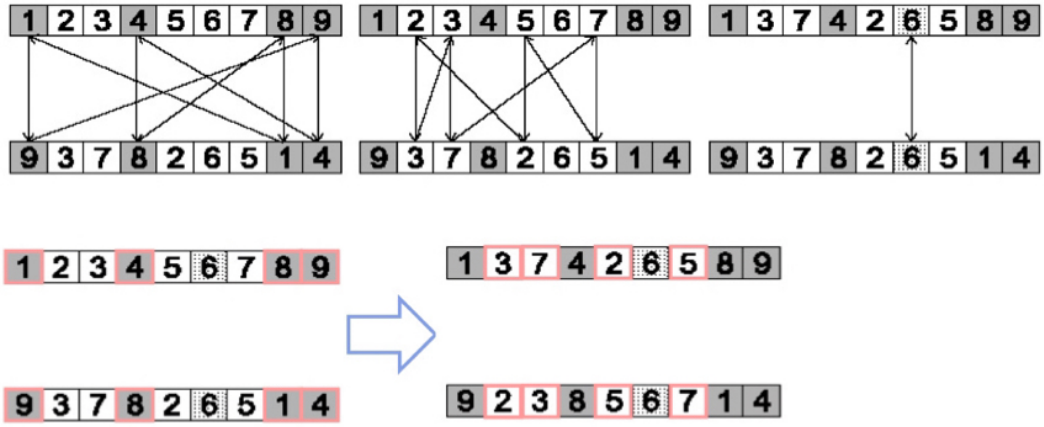
```
// 原本的流程
do {
    for (int j = 0; j < POP_SIZE / 2; ++j) {
        parent_selection(parent);
        crossover(parent, offspring);
        mutation(offspring);
    }
    survivor_selection(offspring);
    offspring.clear();
} while (not exceed the time limit);
```

```
// 後來調整過的流程
do {
    if (a local minimum appears many times) {
        do crossover to generate new offspring
    } else {
        for (int j = 0; j < POP_SIZE / 2; ++j) {
            parent_selection(parent);
            mutation(parent, offspring);
        }
    }
    survivor_selection(offspring);
    offspring.clear();
} while (not exceed the time limit);
```

第一個code在進行剛剛介紹基因演算法的基本環節
第二個code做了一些調整，由於initial population就是基於global placement的結果來greedy擺放instance，所以原先可以藉由instance之間resource的交換（mutation）來達到HPWL下降的目的
但如此可能會陷入local optimum的窘境，所以在一個最優值出現多次後會再用crossover產生出一批純然不同的子代來增加solution space的廣度，之後再繼續用mutation繼續找出較好的solution

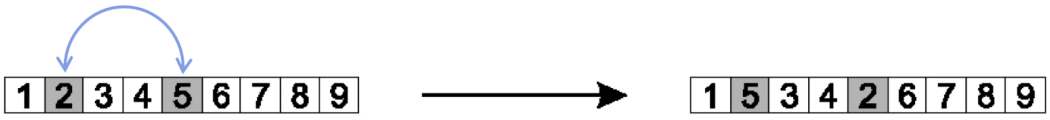
  - **parent_selection()**
    選四個親代並挑選最好的兩個來進行接下來的動作

- **crossover()** 使用的是cycle crossover



1. Make cycles from mapping of alleles between parents
2. Put the alleles of the cycle from P1 in child1
3. Take next cycle from P2 and repeat steps 2~3

- **mutation** 使用的是swap mutation



  - Pick two alleles at random and swap their positions

```
// 以CLB為例
tmp1 = rand() % inst[blockType::CLB].size();
tmp2 = rand() % inst[blockType::CLB].size();
if (incremental_fitness(offspring[offspring.size() - 1],
blockType::CLB, tmp1, tmp2)) {
    std::swap(offspring[offspring.size() -
1].resource_permu[blockType::CLB][tmp1],
offspring[offspring.size() - 1].resource_permu[blockType::CLB]
[tmp2]);
}
tmp1 = rand() % inst[blockType::CLB].size();
tmp2 = rand() % inst[blockType::CLB].size();
if (incremental_fitness(offspring[offspring.size() - 2],
blockType::CLB, tmp1, tmp2)) {
    std::swap(offspring[offspring.size() -
2].resource_permu[blockType::CLB][tmp1],
offspring[offspring.size() - 2].resource_permu[blockType::CLB]
[tmp2]);
}
```

但在這邊我會計算交換了是否會縮小HPWL
如果交換了會縮小HPWL，那才交換；若否，則不交換
判斷是否會縮小HPWL的方法是使用`incremental_fitness()`這個function，它只會計算與交換的兩個instance相關的net的HPWL，而不會計算不相關的net，以此來增加速度

- **survivor_selection()**

```
pool.insert(pool.end(), new_genes.begin(), new_genes.end());
std::sort(pool.begin(), pool.end(), [](const gene& lhs, const
gene& rhs){
    return lhs.fitness < rhs.fitness;
});
pool = std::vector<gene>(pool.begin(), pool.begin() + POP_SIZE);
```

將新產生的子代加入原先的population中並排序
取前POP_SIZE個基因來進入下一輪的演算法

- **output_file()**

```
std::ofstream fout;
fout.open(output_file);
for (size_t i = 0; i < inst[blockType::CLB].size(); ++i) {
    fout << inst[blockType::CLB][i]->name << " " <<
resource[blockType::CLB][pool[0].resource_permu[blockType::CLB][i]]-
>name << std::endl;
}
for (size_t i = 0; i < inst[blockType::RAM].size(); ++i) {
    fout << inst[blockType::RAM][i]->name << " " <<
resource[blockType::RAM][pool[0].resource_permu[blockType::RAM][i]]-
>name << std::endl;
}
for (size_t i = 0; i < inst[blockType::DSP].size(); ++i) {
    fout << inst[blockType::DSP][i]->name << " " <<
resource[blockType::DSP][pool[0].resource_permu[blockType::DSP][i]]-
>name << std::endl;
}
fout.close();
```

將CLB/RAM/DSP instance的名字印出，再將instance對到的resource名字印出

Result

- **Testcase1**

- **Testcase2**

```
fpga-112062554@MakLab:~/final$ time ./legalizer testcase2/architecture.txt testcase2/instance.txt testcase2/netlist.txt output2.txt
Initial HPWL: 1.13176e+07
Final HPWL: 1.13176e+07

real    9m41.775s
user    9m40.439s
sys     0m1.088s
fpga-112062554@MakLab:~/final$ time ./verifier2 testcase2/architecture.txt testcase2/instance.txt testcase2/netlist.txt output2.txt
Reading resource information...
Reading instance information...
Reading netlist information...
Reading output information...
Checking the first constraint...
Checking the second constraint...
Checking the third constraint...
Calculating total HPWL...
Total HPWL = 11317569.5
```

- **Testcase3**

```
fpga-112062554@MakLab:~/final$ time ./legalizer testcase3/architecture.txt testcase3/instance.txt testcase3/netlist.txt output3.txt
Initial HPWL: 256412
Final HPWL: 115001

real    9m45.045s
user    9m1.031s
sys     0m43.967s
fpga-112062554@MakLab:~/final$ time ./verifier2 testcase3/architecture.txt testcase3/instance.txt testcase3/netlist.txt output3.txt
Reading resource information...
Reading instance information...
Reading netlist information...
Reading output information...
Checking the first constraint...
Checking the second constraint...
Checking the third constraint...
Calculating total HPWL...
Total HPWL = 115001.0
```

- **Testcase4**

```
fpga-112062554@MakLab:~/final$ time ./legalizer testcase4/architecture.txt testcase4/instance.txt testcase4/netlist.txt output4.txt
Initial HPWL: 1.0247e+08
Final HPWL: 1.02447e+08

real    9m39.897s
user    9m39.216s
sys     0m0.657s
fpga-112062554@MakLab:~/final$ time ./verifier2 testcase4/architecture.txt testcase4/instance.txt testcase4/netlist.txt output4.txt
Reading resource information...
Reading instance information...
Reading netlist information...
Reading output information...
Checking the first constraint...
Checking the second constraint...
Checking the third constraint...
Calculating total HPWL...
Total HPWL = 102447489.5
```

Learn & Problem

在這個final project中學到蠻多的，原本想要看看有沒有相關的論文可以實作，有找到一篇論文結果做了發現速度非常慢（所以看論文要記得先看experiment result)，所以就放棄第一個方法
之後本來想要用gordian placer+tetris的方法來實作但因為矩陣乘法速度實在太慢所以第二個方法也放棄
再來就是想到自己有修過演化計算的課，覺得可以用到這個project上，雖然最後的成果並沒有很好，但藉由這個過程我覺得我學習到了很多，也比較了解要如何在initial legalization的結果去最大化的進步