HW2 Report

林元泰 109065220

How to compile and execute your program, and give an execution example.

step 1:
Go to src and type the following command,

$ make

step2:
type ../bin/FM cellfile netfile outputfile

example :

```
alanlin@sd234158 src % ../bin/hw2 ../testcases/p2-1.nets ../testcases/p2-1.cells ../output/p2-1.out
Total cell count : 375
Total net count : 357
Parsing cells and nets...
P max = 5

Balancing initial partition...
A set cell count : 188
A set cell area : 286


B set cell count : 187
B set cell area : 287

No need for balancing
Calculating cut size...
cut size = 226

Start FM algorithm...
```

The final cut size and the runtime of each testcase

```
|-----------------------------------------------|
|                                               |
|    This script is used for PDA HW2 grading.   |
|                                               |
|-----------------------------------------------|
grading on 109065520:
  testcase |   cut size |    runtime | status
    pub2-1 |         42 |       0.06 | success
    pub2-2 |        735 |       0.83 | success
    pub2-3 |      13438 |      40.97 | success
    pub2-4 |      48006 |      80.59 | success
    pub2-5 |     127078 |     216.11 | success
    hid2-1 |      76577 |     196.57 | success
    hid2-2 |     102262 |     189.65 | success
    hid2-3 |     143032 |     331.44 | success
|-------------------------------------------------|
|                                                 |
|    Successfully generate grades to HW2_grade.csv|
|                                                 |
|-------------------------------------------------|
```

Where is the difference between your algorithm and FM Algorithm described in class? Are they exactly the same?

There are few differences between my algorithm and FM Algorithm described in class.

1. Searching the cell with max gain in A set and B set:
   In my implementation of the FM algorithm, I don't build the bucket list for A set and B set. Therefore, the method for Searching the cell with max gain in A set and B set is different from the slide.

   I use a for loop (ranged from 0 to num of cells-1) to calculate the index of the cell with max gain in A set and B set. If I get a cell with bigger gain than before, I'll store it and update the biggest gain.

2. Stop condition of FM algorithm:
   It is obvious that you don't have to calculate all rounds of the FM algorithm. In the beginning, the accumulate gain will increase significantly. When the gain of base cells are close to 0, it will reach the best cut size in the specific round. After that, the gain will decrease. The rounds when gain is decreasing is useless and removable. Therefore, I set a stop condition that helps me to reduce the run time.

Did you implement the bucket list data structure?

Yes

I have done some changes to the bucket list. I use a map to replace the linked list since the pointer operation is time consuming. The map includes the name and the pointer of the cell. This design is easy for me to insert and delete the cell of the bucket list.

How did you find the maximum partial sum and restore the result?

I have set the accumulate gain as 0 before starting the FM algorithm. In every round of the FM algorithm, I add the gain of the base cell into the accumulate gain when it is chosen. Before starting to update other cells' gain, I store the base cell into a vector (move_cell) instead of moving it to another set during update. If the accumulate gain after update is bigger than before, I will call it the best gain and record the number of the current round as the best round.

At the time both max gain of cells in A set and B set are negative, I can make sure that the accumulate gain will start decreasing. Therefore, I will stop the algorithm and call the number of the best round which has the best accumulate gain..

With the number of the best round, I start moving the cell recorded in the vector(move_cell) from the first one(index 0) to the one moved during the best round (index best_round).

In the end, I calculate the cut size(using the function I used before) and current set partition into my output file.

Please compare your results with the top 5 students' results from last year and show your advantage either in runtime or in solution quality.

My cut size is larger than the top 5 students' results from last year, there are several reasons why they can generate a better solution.

1. A good initial partition
   A good initial partition is a good start of the FM algorithm. Each initial set partition has a unique sequence of cell set partitions in every round. If there is a pre-calculate implementation before moving cells from B set to A set, it should lead to a good initial partition.

2. A good base cell selection strategy
   When selecting the base cell, it is possible that there are several cells that have the same max gain. It is a critical question to choose which one to be the base cell. This decision will make a huge change to the following sequence of set partitions.

3. A lightweight code
   A lightweight code has faster execution time of the program. By reducing or combining instructions, it is possible to enhance the weight of code.

What else did you do to enhance your solution quality or to speed up your program?

In the base cell selection part, I'll choose the cell that has more pins when I get a cell with the same gain as the biggest gain.

The reason why I choose the cell with more pins is that more connected nets can lead to more possibility to change other cells' gain. For this selection strategy, it is more likely to extend the limitation of cell moving.

What have you learned from this homework? What problem(s) have you encountered in this homework?

In programming part

1. struct v.s. class
   I start with using struct to implement the cell unit and net unit. But at the time I want to assign a net unit in the cell unit, the compiler response that the net unit type hasn't declared before assign.
   I was quite confused at that time. After I chose class to implement the units, this problem was no longer happened.

2. array v.s. vector v.s. map
   I start with using arrays to store units. But after I inserted the data into the array, I found that the size of the array would be not enough or oversized. This will lead to some data loss or memory waste. The most important thing is the size of the array can not be changed after declaration.

   Because of the constraint of array declaration, I choose vector to store units. Everytime I want to store a unit, I push it into the corresponding vector. The size of the vector is changeable, which is a better way than array.

   When I want to select the cell when I get its name, I have to search the vector from 0 until I find the same name in index i. This search consumes too much time. So I create a map to solve this problem.

   I use the cell's name as indexes of the map. When I want to find the corresponding cell, I just use the name as the index and find the pointer of the cell. It is much faster than searching the whole vector.

In structure construction part

1. Bi-direction linked list
   It has been a long time since the last time I implemented the linked list. So I have done some search on it.

2. Pointer addressing
   In the beginning, I store data in the unit then push it into a vector. At the time I wanted to create a map whose indexes are the cell's name, I found that it is impossible to share the same data between two data structures.

   I create a pointer of the unit and use it to new a unit in the memory. Later, I create a map which type is also a pointer and point it to the same address of memory. In the end, Push both pointers into vector and map respectively.

3. Timing analysis
   After finishing the coding, I test my program with test cases attached in the spec. I found that the execution time of the FM algorithm is longer than I thought. I start testing the execution time of each part and modify the structure such as reducing the times of calling high time complexity functions.