CS6135 VLSI PDA HW2

111062684　林鼎勳

1) How to compile and execute your program, and give an execution example:
   - Compile: Go into the directory “HW2/src/” and enter the command:
     `$ make`
     This will generate the executable file “hw2” in the directory “HW2/bin/”.
   - Execute: In the same directory (“HW2/src/”) and enter the command:
     `$ make tc_1`
     This will execute "hw2" and read "p2-1.cells" and "p2-1.nets" under the directory "../testcases/" as input data and write the result into "p2-1.out" in the directory "../output/".
     You can replace `tc_1` with `tc_2` to `tc_5` to execute and produce the result of each testcase.

2) The final cut size and the runtime of each testcase:
   I test each testcase for 5 times and average the result.

Testcase	1	2	3	4	5
Cut Size	238	2342	23017	81520	163236
Runtime (sec.)	0.056s	0.171s	7.624s	32.199s	77.314s

3) Runtime = T_{IO}+$T_{computation}$. For each case, please analyze your runtime and find out how much time you spent on I/O and how much time you spent on the computation (FM Algorithm):
   I test each testcase for 5 times and average the result.

Testcase	1	2	3	4	5
T_{IO}	0.030s	0.070s	0.468s	1.506s	2.957s
$T_{computation}$	0.026s	0.101s	7.156s	30.693s	74.357s
Total	0.056s	0.171s	7.624s	32.199s	77.314s

4) The details of your implementation containing explanations of the following questions:
   a) Where is the difference between your algorithm and FM Algorithm described in class? Are they exactly the same?
      No, they are not the same. I add a set of new movements before the movement in the FM Algorithm. You can just think of it as a preprocessing step to improve the performance in timing.
      The detail of the preprocessing step is described below:

0. First compare the area of the two groups. Here we assume group A is larger than group B.
1. Move the cell whose gain value is the biggest in group B to group A if the movement meets the area constraint. If not, skip it and try the next cell instead. After the movement, lock the cell. (Noticed that the larger area group will become larger, and vice versa.)
2. Repeat 1. until the biggest gain value is smaller than -1.
3. Move the "free" cell whose gain value is the biggest in group A to group B if the movement meets the area constraint. If not, skip it and try the next cell instead. After the movement, lock the cell.
4. Repeat 3. until the biggest gain value is smaller than -1.
5. Repeat 1. to 4. until the cut size is not decreasing any more.

By adding this preprocessing step, it can reduce the total runtime and perform similar results. The following table shows the differences in timing and cut size. (Take testcase 4 & 5 for example, and I test each testcase for 5 times and average the result.)

	Testcase 4		Testcase 5	
	Runtime	Cut Size	Runtime	Cut Size
Without *	38.698s	81471	252.665s	163092
With *	32.199s	81520	77.314s	163236
Difference	↓16.794%	↑0.060%	↓69.401%	↑0.088%

*: preprocessing step

The result shows that the timing improvement is significant and the cut size difference is within 0.1%.

b) Did you implement the bucket list data structure?

No. But I implemented a similar data structure to store and edit the gain value. The data structure to store gain value looks like this:

```
map<int, unordered_set<string>, greater<int>>
```

I use `map` because it can help me order the gain value from large to small, and the `unordered_set` inside can fasten the search time.

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

In my code, I don't really care about the maximum partial sum because while establishing the gain value table, I only store the cell whose gain value is larger than or equal to -1. But with this property, I still use some temporary data structures to store the result of each iteration. This is

because I allow the minimum gain value to be -1, which means the cut size might increase. So, in order to get the best result, I use the temporary data structure to store the last iteration result and compare with the current iteration. If the current iteration is better, I store it in the temporary data structure to compare with the next iteration. If the last iteration result is better, I just break out the loop and terminate.
Another concern you may arise is why I set the minimum gain value to -1. This is because in the FM Algorithm, we may trap in a local minimum. To escape from the current local minimum, we need to add some bad moving to let it have the opportunity to get away from the current local minimum and hope to find another better local minimum.

d) What else did you do to enhance your solution quality (you are required to implement at least one method to enhance your solution quality) and to speed up your program?

1. In 4.a., I implement a preprocessing step to speed up the program and remain almost the same performance.
2. In 4.c., I allow the movement of a cell whose gain value is -1. This will give some chances to escape from the local minimum and find another better local minimum.
3. I use lots of `unordered_set` and `unorderend_map` in my code to fasten the search time.
4. I store the gain value table for each group separately to enhance the timing. Also, I discard the cell whose gain value is less than -1. This helps us to save space and improve performance while maintaining the gain value table.
5. In the cell moving step, which is right after the preprocessing step, I'll first move the cell in the larger area group to the smaller area group. This is trying to remain a balance between the two groups, and this will help to increase the iterations of moving the cell and reduce the probability of terminating the process because of area constraints.
6. I do not save the whole cut nets. Instead, I just save the number of cut nets. This is easy to implement because I have a table to save the number of the front side and the to side of each net. To calculate the cut size, I just check whether both sides of a net are not 0.

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

In this homework, I got more familiar with C++ and its containers. Besides, I understood the whole process of the FM Algorithm, and know how hard to get the best result in a real-life case while keeping a good performance no matter in space or timing.

There are two problems I have encountered with this homework. The first one is the data structure. I took a long time to decide which data structure to use and how to combine them to improve and simplify my implementation. The second one is the timing. In the early version of my code, I used `multimap` to store and edit the gain value of each cell. The result was terribly bad. This is because in each deletion or insertion, I need to go through a red-black tree (or binary search tree). It took a very long time when the size of the data was large. To fasten my code, I first try using `forward list` inside the `map`. But this didn't work either. Last, I replace `forward list` with `unordered set` inside the map, and this works! With this data structure, I can still have an ordered gain value table while decreasing the map size, and also fasten the search time, insertion time, and deletion time while editing the gain value table.