Physical Design PA1 report

Data structures

1. Bucket list:
Our bucket list is saved as **map<int, Node*>** type. where the key is the gain of cells, and the value be cell. In addition, the cells with the same gain is sorted in a double-link list structure.

Algorithms

The most important algorithms in FM-heuristic is “update gain” and “find max gain cell”.

1. update gain:
The pseudo code of update gain is as follow:

```
Algorithm: Update_Gain
begin /* move base cells and update neighbors' gains */
F <- the From Block of the base cell;
T <- the To Block of the base cell;
Lock the base cell and complement its block;
for each net n on the base cell do /* check critical nets before the move */
    if T(n) = 0 then increment gains of all free cells on n
    else if T(n) = 1 then decrement gain of the only T cell on n, if it is free
    /* change F(n) and T(n) to reflect the move */
    F(n) <- F(n) - 1; T(n) <- T(n) + 1;
    /* check for critical nets after the move */
    if F(n) = 0 then decrement gains of all free cells on n
    else if F(n) = 1 then increment gain of the only F cell on n, if it is free
end
```

Notice that either ”increment or decrement on all free cells” or ”increment or decrment on the only free cell” requires a complexity of the size of net. We might think that if we run this algorithm per iteration, then it might get $O(P^2)$ complexity, since each net requires quadratic time complexity. However, the update gain function is only $O(P)$. We can show that the 4 condition in the update gain function will only be run at most one time!!! Since once a cell is moved, it is locked throughout the iteration, we can easily see that the 4 condition won’t happen more than twice. The worst condition is that for a Net, its initial (From/To) size is (n/0), then (n-1/1), …, (1,n-1),(0,n), so the 4 conditions are all runned. Therefore, the overall complexity of update gain is $O(P)$ per iteration.

2. find max gain cell:
Since we maintain a key structure for the gain. Directly access the max gain cell in the bucket list is O(1). However, we need to check if the max gain cell can balance the partition after move. If we just simply iterating over the map structure then it might have $O(P_{max})$ complexity, and will be even worst for running n_{cell} times.
We can find a better solution in this problem since all cells have unit size. For both part, we can calculate if moving any cell from one side will balance the partition in $O(1)$ complexity, then we find max gain in the side that is valid for balance partition. This required only $O(1)$ per step, and $O(n_{cell})$ Complexity.

Discussions

1. The overall complexity is $O(P \times n_{iterations})$

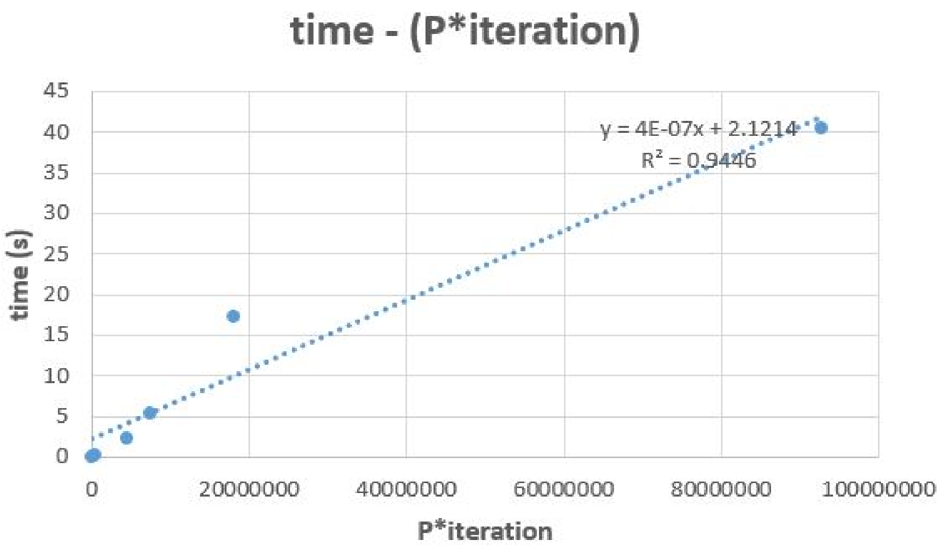
Apart from the "update gain" and "find max gain cell" algorithm, other operations is obvioulsy $O(P)$ complexity. And we have shown in the previous section that both these two functions are also in linear time per iteration. Therfore, the total complexity of the overall FM-algorithm might be $O(P \times n_{iterations})$.

Results

The following table shows the input size for each testcase and the reun time for running our algorithm.

$input\ file\ name$	$n_{terminals}$	$n_{iterations}$	$n_{terminals} \times n_{iterations}$	$t_{exe}(sec.)$
input_0.dat	721451	25	18036275	17.300
input_1.dat	12496	9	112464	0.045
input_2.dat	24928	15	373920	0.12
input_3.dat	266821	17	4535957	2.259
input_4.dat	501426	15	7521390	5.327
input_5.dat	1451278	39	92881792	40.556

The following figure shows that run time and $n_{terminals} \times n_{iterations}$ are strongly and linearly correlated. So the experiment is consistent with our complexity analysis.



Reference:

[1] A Linear-Time Heuristic for Improving Network Partitions, C.M. Fiduccia and R.M. Mattheyses, 19th Design Automation Conference.