CAD PA2 Report

link:
https://hackmd.io/@unj0M9DkQhqZGOyd71BT5g/SkfpC_fD9 (https://hackmd.io/@unj0M9DkQhqZGOyd71BT5g/SkfpC_fD9)

(1) Name and student ID

Name: 許智凱
Student ID: 108061239

(2) Execution of the program

Please just follow the commands given by TA.

```
$ cd CS3130_PA1
$ make
$ ./pa2 <.in file> <.out file>
(e.g., $ ./pa2 testcases/basic/case00. in case00.out)
```

Note that argv[1] and argv[2] are the file_path of the input file and the output file.

(3) Data structure and the algorithm

We basically follow the concept of FM algorithm. Here is the class we design.

```
class FM_algorithm
{
public:
    FM_algorithm() {}
    void read_input_data(string file_path);
    void gen_output_file(string file_path) const;
    void solve();
    void print() const;
    void print_step_list() const;
private:
    /* private data*/
    int max_area;
    int n_cells;
    int n_nets;
    int total_size = 0;
    int k;                       // k-way
    int G_max;                   // max partial sum of gain (initialize in each iteratio
    int nth_step = 0;
    int nth_ite = 0;
    int evaluated_cost;
    vector<Cell> cell;
    vector<Net> net;
    vector<int> gain;            // the gain of each cell (initialize in each iteration)
    vector<int> cell_gr;         // the group of each cell (initialize in each iteration
    vector<int> locked_cell;     // the locked cell (initialize in each iteration)
    vector<int> partition_size;
    vector<vector<int>> net_distri;
    list<Step> step_list;        // log of each step (initialize in each iteration)
    /* private function*/
    void judge_way();
    void construct_cell_array();
    void initial_partition();
    void compute_initial_info();   // net_distri, gain, partition_size
    int choose_base_cell();
    void move_and_update();
    int evaluate() const;
};
```

For the implementation, we use `vector`, `list`, `pair`. Additionally, we define three classes to assist the algorithm, `Cell`, `Net`, and `Step`.

Cell

```
class Cell {
friend class FM_algorithm;
public:
    Cell() { n_pins = 0; size = 0; group = -1; }
private:
    int n_pins;
    int size;
    int group;
    vector<int> N;      // the nets it connected
};
```

It describes the information of a cell, including how many pins it has, the size, the group it belongs to, and the nets it connects to.

Net

```
class Net {
friend class FM_algorithm;
public:
    Net() { n_pins = 0; }
private:
    int n_pins;
    vector<int> C;      // the cells it connected
};
```

It describes the information of a net, including how many pins (cells) it connected to, and the cells it connects to.

Step

```
class Step {
friend class FM_algorithm;
public:
    Step();
private:
    int move_cell;
    int F;              // from partition
    int T;              // to partition
    int g;              // gain
    int G;              // partial sum of gain
    int balancy;        // new_T_cell_size - total_size / k, the smaller the better
};
```

`Step` is to store the information of each moving (in one iteration). The info includes the cell moved in this step, from which group (partition), to which group (partition), the gain, the partial sum of gain, and the balancy, which is defined as

$$x = \frac{NewToCellSize - TotalSize}{k}.$$

The main structure of the entire algorithm is in the function `solve()` . After judging the ways (2-way or k-way) by `judge_way()` , we first do `initial_partition()` . Our algorithm mainly focuses on 2-way. For k-way, we directly finish the partition in `initial_partition()` and evaluate. Then we basically follow the FM algorithm. In the following, we will introduce some important part and importants variables.
First, we `construct_cell_array()` . This is to find out the `n_pins` of each cell, which will be helpful for later algorithm.
At the starting of each iteration, we perform `compute_initial_info()` . It computes the `net_distri` , `gain` , and `partition_size` .

- `vector<vector<int>> net_distri` : It stores the distribution of cells in different groups, in whcih the cells are connected by net i. For
- `vector<int> gain` : The initial gain of each cell.

We perform `choose_base_cell()` as the first movement, whcih decides the base cell that's going to be moved.
Then we perform `move_and_update()` and

`chooose_base_cell()` repeatly. It terminates until all the cells are locked or there is no cells that can be moved without violating the area constraint.

CHOOSE_BASE_CELL()

To find out the candidates of base cells efficiently, we construct a new gain vector and sort it descendingly by the gain values. But the constrction and sorting need to be performed each time this function runs, whcih is not efficient. We think there must exist a better approach.
The "candidates" means all of these cells are with the same gain. Then we choose the candidate cells easily by sorted gain vector.
The criterion to choose a base cell is not by the balancy in Prof. Wang's lesson. Here we choose the cells with larger `n_pins`, which is computed in `construct_cell_array()`.
By experiment this seems have a better performance.
Then we write this movement into the `list<Step> step_list`.

MOVE_AND_UPDATE()

In this part, we update `gain` before the movement. Move. Update the `gain` after the movement.

After the loop terminates, we decides the movements by `G_max`, the maximum of partial sum, and the `step_list`.
Then update the new partiotion. The above is an iteration.
Iterate until `G_max` < 0.

(4) other details of the implementation

A. The initail partition

We find out the initial partition significantly affects the mincut of partition result. Originally, I just cut half. One half are gived to partition 0, the other half are gived to partition 1.
Later, we changes the way. We put the cells connected by the same net as together as possible. To do this, we extrally compute the sum of cell area that a net connected and sort them.

The result is very good. For test case00, the cost changes from 1954 to 1134.

B. The choice of base cell

Rather than choosing by balancy, we choose by number of pins. This improves a little. We guess this is because the two reasons:

1. The earlier a cell be decided, the larger impact on the final result it has. (From A, we can see that,)
2. More nets a cell connected, larger impacts the mincut.

We think due to the two reasons, this approach has a better performance.