How to Compile & Run

There are serveral details in my F-M implementaion:

1. I randomly partition a given circuit and apply F-M to improve cut size.
2. If enable multiple passes, my algorihm will keep running until improvement ratio is less than 5% or the number of passes is 10.
3. Otherwise I will run F-M for only one pass.

To compile F-M partitioning algorithm, you need GNU C++ Compiler at least v7.0 with -std=c++17. I recommend out-of-source build with cmake:

```
~$ mkdir build
~$ cd build
~$ cmake ../
~$ make
```

You will see an executable file `fm` under `bin/`. To run F-M, you can simply type:

```
~$ cd bin
~$ ./fm [input file name] [output file name] [1/0 to enable multiple passes
```

For example, to enable multiple passes:

```
~$ ./fm input_pa1/input_0.dat output_0.dat 1
```

Experimental Results

I implement F-M using C++17 and compile F-M using GCC-8 with optimization -O3 enabled. I run F-M (**single CPU core**) on twhuang-server-01

Single pass

The following table shows runtime of my F-M implementation on each benchmark for one pass. My F-M implementation can finish all benchamrks **within 10 seconds**. In input*0.dat, the*

*algorithm requries 9 seconds to finish. That is because input*0.dat contains the largest number of cells, which induces noticable overhead of updating buckets at each iteration within a pass.

Input	cutsize	runtime
input_0.dat	42256	9.00s
input_1.dat	2508	0.02s
input_2.dat	4247	0.04s
input_3.dat	35008	0.40s
input_4.dat	97944	0.89s
input_5.dat	288264	2.96s
input_6.dat	2	0.004s

Multiple passes

The following table shows runtime and cut size of our F-M on each benchmark for multiple passes. My algorithm will keep running F-M until the improvement ration is less than 5% or the number of passes is 10. We can clearly see my implementation can get signifcant improvement after several passes. Take input_0.dat as an example, the cut size using my F-M implementation is **6.8x** compared to initial partition.

Input	initial cut_size	final cut_size	# passes	runtime
input_0.dat	109154	16151	7	49.7s
input_1.dat	3062	1559	6	0.03s
input_2.dat	6241	2690	5	0.05s
input_3.dat	62995	28287	3	0.56s
input_4.dat	118491	58274	7	1.68s
input_5.dat	342611	173204	6	5.02s
input_6.dat	2	2	1	0.004s

Challenges

During the F-M implementation, I encounter three challenges:

1. The cut size cannot converage for multiple passes.
2. The overhead of inserting/erasing cells into bucket lists is large.
3. The overhead of updating gain of each cell is large.

Apparently, the fist challenge is due to a bug in my code. To identify where the bug is, I carefully go through a simple testcase (i.e., input_6.dat) and print out the updated gain of each cell at each iteration within a pass. The print-out results show I did not correctly update the gain of each cell. For the second and thrid challengs, I dive into the F-M paper and figure out the implemntation details.