CS6135 VLSI Physical Design Automation

Homework 5: Global Routing

Due: 23:59, January 21 2021

1. Introduction

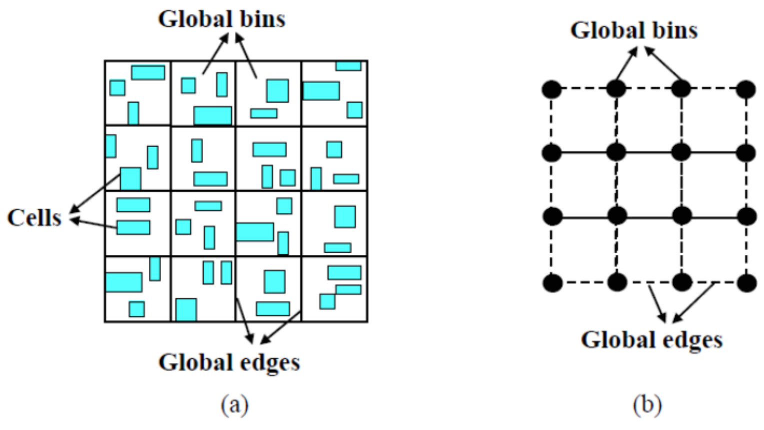
Routing is a crucial step in IC design. The placement stage determines the location of each cell of an IC design. Then in the routing stage, we need to connect all the placed cells with wires properly. The routing problem can be divided into two steps: global routing and detailed routing. During global routing, nets are connected on a coarse-grain grid graph with capacity constraints. Then detailed routing follows the solution obtained in global routing to find the exact routing solution. The quality of global routing affects timing, power and density in the chip area, and thus global routing is a very important stage in the design cycle.

In this homework, you are asked to implement an existing algorithm or develop your own algorithm to solve the global routing problem with a set of 2-pin nets.

2. Problem Description

(1) Input:

- The layout is partitioned into rectangular grids called global bins, as known as G-cells, as shown is Figure (a), and each pin is assumed to lie at the center of the gird that contains the pin.
  In the grid graph G shown in Figure (b), each vertex $v \in V$ represents a global bin, and each edge $e \in E$ corresponds to a boundary between two adjacent global bins.
- Vertical and horizontal capacities of grids' boundaries.
- A set of nets, $N = \{n_1, n_2, \ldots, n_k\}$, to be routed over a grid graph $G = (V, E)$. Each net $n_i, 1 \leq i \leq k$, has a set of pins to be connected.



(2) Output:

The critical points' coordinates of each net after global routing is done. Notice that "critical points" means the starting point, ending point and turning points.

(3) Objective:

The routing problem for a net n_i is to find an additional subset of vertices, $s_i \subseteq V$, and a subset of edges, $E_i \subseteq E$, to form a Steiner tree $T_i = (V_i, E_i)$, where $V_i = n_i \cup s_i$. The supply $s(e)$ of an edge e represents the number of available routing tracks passing through it. The number of wires that utilize an edge e is called the demand $d(e)$ on the edge. In other hand, the demand $d(e)$ represents how many nets that pass through e. The overflow on an edge e is defined to be the difference between its demand and supply as shown below:

$$overflow(e) = \begin{cases} d(e) - s(e), \text{if } d(e) > s(e) \\ 0, \text{otherwise} \end{cases} \tag{1}$$

The major optimization objective of global routing is to minimize the total overflow on all edges as shown in (2), and the second objective is to minimize the total wirelength on all edges as shown in (3).

$$\min\left(\sum_{\forall e \in E} overflow(e)\right) \tag{2}$$

$$\min\left(\sum_{\forall n_i \in N} \sum_{\forall e \in E_i} length(e)\right) \tag{3}$$

This homework asks you to write a global router that can route multiple nets. To simplify the global routing problem, we have some simplifications as follows:

(a) Consider only two layers (vertical and horizontal).

(b) Consider only grid-based coordinates. The lower left corner of the global routing region in each layer is (0, 0). The width and height of grids are ignored because all x- and y-coordinates are grid-based. As the result, $length(e)$ will always be 1.

(c) Consider only 2-pin nets.

(d) Consider only fixed wire width and spacing. All wire width and spacing are equal to 1.

3. Input Format

There is only an input file of each testcase. Here is an example:

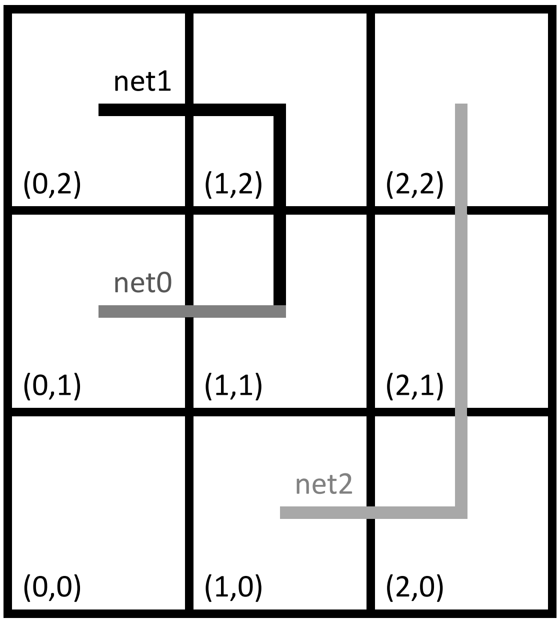
```
grid 3 3
// grid # of horizontal grids # of vertical grids
vertical capacity 2
// vertical capacity vertical capacity
horizontal capacity 2
// horizontal capacity horizontal capacity
num net 3
// num net # of nets
net0 0 2
// net-name net-id # of pins
   0 1
// pin x-grid coordinate pin y-grid coordinate
   1 1
net1 1 2
   0 2
   1 1
net2 2 2
   2 2
   1 0
```

4. Output Format

Here is an example:

```
net0 0
// net-name net-id
(0, 1, 1)-(1, 1, 1)
// (pin x-grid coordinate, pin y-grid coordinate, 1)
!
net1 1
(0, 2, 1)-(1, 2, 1)
(1, 2, 1)-(1, 1, 1)
!
net2 2
(2, 2, 1)-(2, 0, 1)
(2, 0, 1)-(1, 0, 1)
!
```

Example:



5. Language/Platform

(1) Language: C/C++

(2) Platform: Unix/Linux

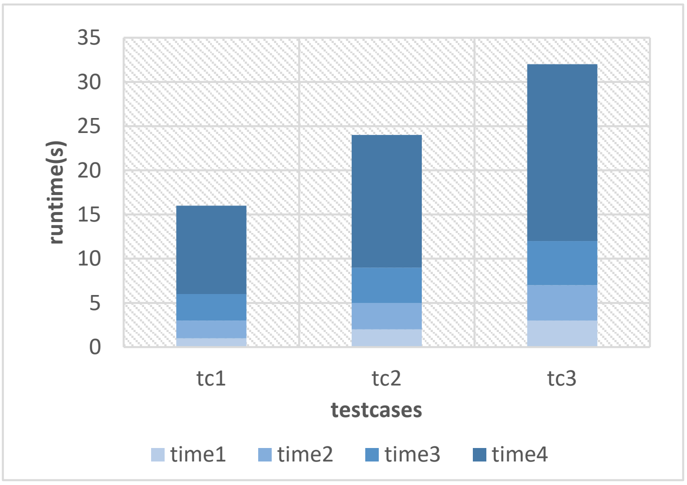
6. Report

Your report must contain the following contents, and you can add more as you wish.

(1) Your name and student ID

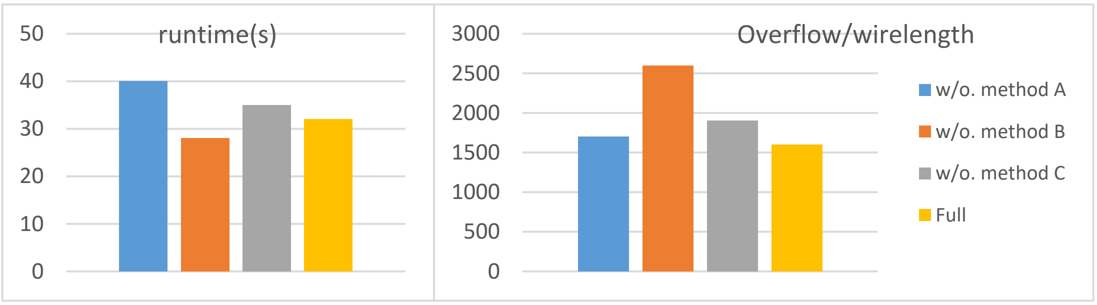
(2) How to compile and execute your program and give an execution example.

(3) The total overflow, the total wirelength and the runtime of each testcase. Notice that the runtime contains I/O, constructing data structures, computing parts, etc. The more details your experiments have, the more clearly you will know where the runtime bottlenecks are. You can plot your results like the one shown below.



(4) The details of your implementation. You have to use flow chart(s) to help elaborate your algorithm, and please follow the symbols usually used in flow charts. (If you are not familiar with the symbols, please refer to this reference: https://www.programiz.com/article/flowchart-programming)
If your method is similar to some previous works/papers, please cite the papers and reveal the difference(s).

(5) What tricks did you do to speed up your program or to enhance your solution quality? Also plot the effects of those different settings like the ones shown below.



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

7. Required Items

Please compress `HW5/` (using tar) into one with the name `CS6135_HW5_${StudentID}.tar.gz` before uploading it to iLMS.

(1) `src/` contains all your source code, your `Makefile` and `README`.
   - ➢ `README` must contain how to compile and execute your program. An example is like the one shown in HW2.

(2) `output/` contains all your outputs of testcases for the TA to verify.

(3) `bin/` contains your executable file.

(4) `CS6135_HW5_${StudentID}_report.pdf` contains your report.

You can use the following command to compress your directory on a workstation:

```
$ tar -zcvf CS6135_HW5_{StudentID}.tar.gz <directory>
```

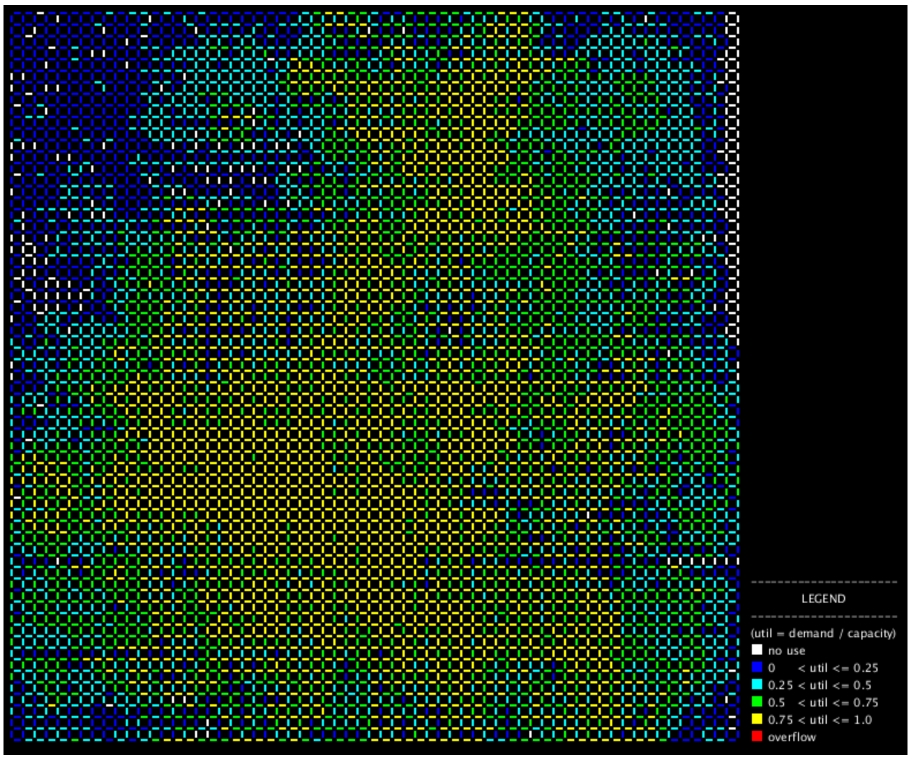
For example:

```
$ tar -zcvf CS6135_HW5_109062501.tar.gz HW5/
```

8. Grading

- ✓ 80%: The solution quality (total overflow and total wirelength) and the runtime of each testcase; hidden testcases included.
- ✓ 20%: The completeness of your report.
- ✓ **5% Bonus**: Parallelization. Please specify your system specification.

✓ **5% Bonus**: For each testcase, you could draw the congestion map like the following example. The congestion map and the legend are needed.



Notes:

- Make sure the following commands can be executed.
  - Go into directory "`src/`", enter "`make`" to compile your program and generate the executable file, called "`hw5`", which will be in directory "`bin/`".
  - Go into directory "`src/`", enter "`make clean`" to delete your executable file.
- Please use the following command format to run your program.

  ```
  $ ./hw5 *.modified.txt *.result
  ```

  E.g.:

  ```
  $ ./hw5 ../testcase/ibm01.modified.txt ../output/ibm01.result
  ```
- Use arguments to read the file path. Do not write file path in your code.
- Program must be terminated within 10 minutes for each testcase.
- Grading is based on total overflow (primary), total wirelength (secondary) and runtime (tertiary).
- We will test your program by shell script. Please make sure your program can be executed by HW5_grading.sh.