Optional Programming Assignment #2: Global Routing (due 6pm, April 28, 2019 on-line)

Simplified from the ISPD'07 and ISPD'08 Global Routing Contests

This programming assignment asks you to write a global router, i.e. to determine the tile-to-tile routing path for each net and minimize the total overflow. To simplify the problem, we have some simplifications for the problem as follows:

1. We consider only two layers (layer 1 is for horizontal routes and layer 2 is for vertical ones).
2. We consider only fixed wire width and spacing. In our test cases, all wire widths, wire, and via spacing are equal to 1.
3. All pins are located in layer 1.

1. Input/Output Specification

Input Format

The file format for the global routing contest is illustrated with comments in italics (these comments will not be in actual input files) and numbered by "#" as shown below:

grid # # 2 *(number of horizontal tiles, vertical tiles, and routing layers)*
vertical capacity 0 # *(the default vertical capacity for each layer)*
horizontal capacity # 0 *(the default horizontal capacity for each layer)*
minimum width 1 1 *(the wire width for each layer)*
minimum spacing 1 1 *(the wire spacing for each layer)*
via spacing 1 1 *(the via spacing for each layer)*
lower_left_x lower_left_y tile_width tile_height *(the lower left coordinates of the chip and the width and height of global tiles)*

num net # *(number of nets for global routing)*
net_name net_id number_of_pins 1 *(net name, net id, # of its pins, its width)*
x y 1 *(the x, y, and layer coordinates of pin)*
x y 1 *(the x, y, and layer coordinates of pin)*
... *(repeat for the appropriate number of pins)*
(repeat for the appropriate number of nets)

*(number of capacity adjustments to modified the default capacity of layers)*
column row layer column row layer new_capacity_value
… *(repeat for the number of capacity adjustments)*

The first line **grid** gives the problem size in terms of the number of horizontal and

vertical tiles and routing layers. Each global routing tile (tile in short) has a *capacity* on its four boundaries to measure the available space. The default capacity value of each layer is specified by the **vertical capacity** and **horizontal capacity** lines. For example, "vertical capacity 0 4" denotes that the default vertical capacity of layer 1 is 0, and that of layer 2 is 4.

The **minimum width**, **minimum spacing**, and **via spacing** lines give the width of a wire, wire spacing between wires, and via spacing between vias for each layer. All of them are equal to 1 in this problem. (Note that we do not need via spacing in this problem). The maximum number of routing paths passing through a tile boundary with the capacity *c* is calculated by *c/(w+s)*, where *w* and *s* represent the wire width of a wire and the spacing between wires, respectively. Because in this problem both *w* and *s* are equal to 1, the tile boundary with capacity *c* can accommodate *c/(1+1) = c/2* routing paths. For example, a tile boundary with capacity 10 can accommodate 5 routing paths.

lower_left_x, lower_left_y, tile_width, and **tile_height** give the lower-left coordinate of the chip and the width and height of global tiles, respectively. Then the nets required to route are given. For example, the statement below gives two nets *A* and *B* for global routing, where the net IDs of A and B are 0 and 1, respectively. Net *A* has 2 pins located in (5,8,1) and (15,8,1), whereas *B* has 3 pins located in (5,5,1), (13,13,1), and (25,15,1).

```
num net 2
A 0 2 1
   5 8 1
   15 8 1
B 1 3 1
   5 5 1
   13 13 1
   25 15 1
```

Finally, the capacity adjustments, specified by the **column row layer column row layer new_capacity_value** format, are given to modify the default capacity value of a specific tile boundary between two adjacent tiles. For example, below shows two capacity adjustments which change the capacity of the tile boundary between tiles (0,0,2) and (0,1,2) to 0, and the capacity of the tile boundary between tiles (0,0,1) and (1,0,1) to 8.

```
2
0 0 2 0 1 2 0
0 0 1 1 0 1 8
```

Output Format

The output file format is as follows:

[net_name] [net_id] [# of routes]
([x11],[y11],[z11])-([x12],[y12],[z12])
([x21],[y21],[z21])-([x22],[y22],[z22])
...
!
(repeat for an appropriate number of nets)

Note that each coordinate is specified by the center coordinate of a global tile (not the original pin coordinate). For example, if the lower-left coordinate of a chip is (0,0) and both the width and height of tiles are 10, then the net *A* in the above example has two pins located in tiles (0,0,1) and (1,0,1) and the resulting path should be (5,5,1)-(15,5,1), instead of (5,8,1)-(15,8,1) (since the center coordinate of tile (0,0,1) is

(5,5,1), and the center coordinate of tile (1,0,1) is (15,5,1)).

Also note that the path in the output should only be horizontal lines in layer 1, vertical lines in layer 2, or via connections between layers 1 and 2. For example the diagonal path (15,65,1)-(25,55,1) is not correct, while (5,25,2)-(5,5,2) and (5,15,1)-(5,15,2) are correct.

2. Problem Statement

Given the routing area, design rules, and a netlist, a global router connects all nets by tile-to-tile paths. The main objective is to minimize the total number of overflows, and the second objective is to minimize the total tile-to-tile wirelength. Here, the overflow on a tile boundary is calculated as the number of routing paths that exceeds the given capacity *c*, *i.e.*, overflow = max(0, # of routing paths - *c*/2). A tile-to-tile wirelength of a net is calculated as the sum of the number of crossed boundaries and the number of vias. Below shows a sample input and output for the instance in Fig. 1.

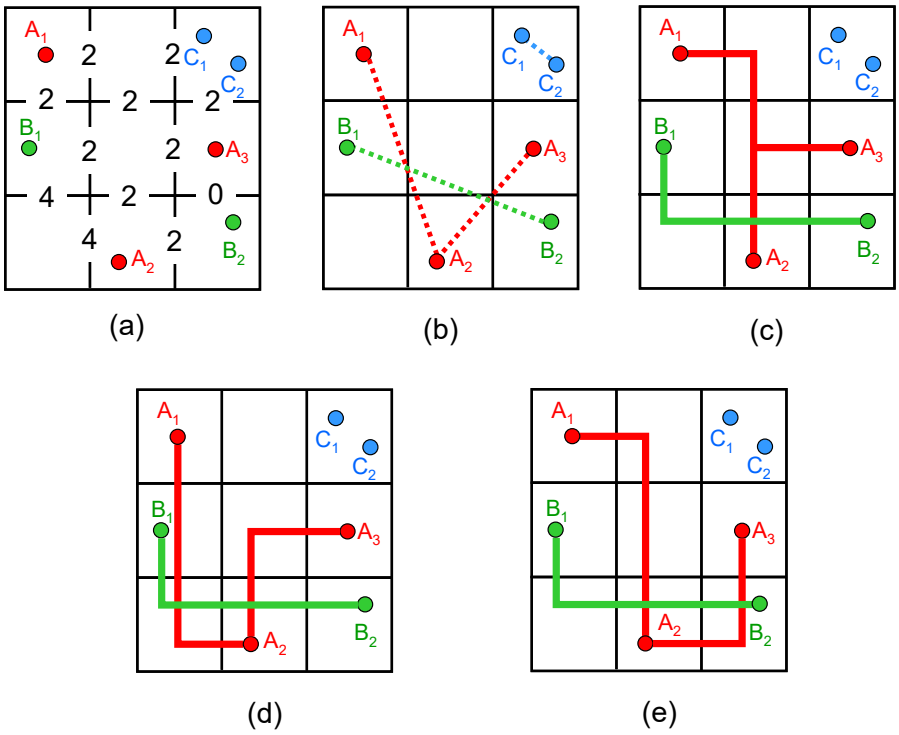

Fig. 1: A routing problem and its possible solutions. (a) A 3x3 routing instance with three nets *A*, *B*, and *C*. The horizontal capacities of tile boundaries on layer 1 and vertical capacities of tile boundaries on layer 2 are shown in the aerial view. (b) The net decomposition for each net. (c) A good routing solution with total overflow = 0 and wirelength = 12, where the wirelength of net A is contributed by 4 crossed boundaries ((0,2,1)-(1,2,1) , (1,2,2)-(1,1,2), (1,1,2)-(1,0,2), (1,1,1)-(2,1,1)) and 3 vias ((1,2,1)-(1,2,2), (1,1,2)-(1,1,1), (1,0,2)-(1,0,1)). (d) A routing solution with total overflow = 0 and wirelength = 14. (e) An inferior routing solution with total overflow = 2 and wirelength = 14.

Sample input file (Fig. 1(a)):

```
grid 3 3 2
vertical capacity 0 2
horizontal capacity 2 0
minimum width 1 1
minimum spacing 1 1
via spacing 1 1
0 0 10 10

num net 3
A 0 3 1
   5 25 1
   13 3 1
   25 15 1
B 1 2 1
   3 15 1
   27 7 1
C 2 2 1
   23 27 1
   27 22 1

3
0 0 1 1 0 1 4
0 0 2 0 1 2 4
2 0 2 2 1 2 0
```

Sample output file (Fig. 1(c)):

```
A 0 6
(5,25,1)-(15,25,1)
(15,25,1)-(15,25,2)
(15,25,2)-(15,5,2)
(15,5,2)-(15,5,1)
(15,15,2)-(15,15,1)
(15,15,1)-(25,15,1)
!
B 1 4
(5,15,1)-(5,15,2)
(5,15,2)-(5,5,2)
(5,5,2)-(5,5,1)
(5,5,1)-(25,5,1)
!
```

(Note that it does not need to output the net *C* because it is a local-tile net.)

3. Hints

You can first model the routing problem as a graph where each node represents a global tile and each edge denotes the tile boundary between adjacent tiles. The cost of an edge could be set to reflect the capacity usage (*e.g*, edge cost = demand/capacity). Then this problem can be solved by Dijkstra's shortest path algorithm. Note that different edge costs would result in different routing results; for example, you can also model the edge cost as $2^{(\text{demand/capacity})}-1$.

The TA has provided the input parser and the net decomposition codes (by the minimum spanning tree algorithm) for you. In addition, you can run the verification script "eval2008.pl" provided by the ISPD'08 global routing contest to verify you routing results.

Because the test case could require a large memory (may > 2G) to run, you might need to test you programs on the linux servers at the EDA Union Lab.

4. Required Files

You need to submit the following materials in a .tar.gz or .zip file:

(1) Source codes (e.g. router.cpp);
(2) Executable binary;
(3) A text README file describing how to compile and run your programs.

The submission filename should be <student id>-pa2.tar.gz or <student id>-pa2.zip (e.g. r06901000-pa4.zip). If you have a modified version, please add –v [version number] as a postfix to the filename and resubmit it to the submission website (*e.g.*, r06901000-pa2-v1.zip, r06901000-pa2-v2.zip, etc.).

Please do *NOT* attach the resulting output files (because they could be too big to be uploaded).

5. Language/Platform

- Language: C, C++, or Java.
- Platform: The machines at the EDA Union Lab. Because the file size of this programming assignment could be very large, **all submitted programs that cannot be successfully run on the machines would incur significant penalties.**

6. Evaluation

The individual score per test case is determined by the total overflows of your global routing solution. A tie is broken by the routed total wirelength. Each benchmark should be complete within 3 hrs on any machine in the EDA Union Lab. Please set the time limit for your program.

7. Resources

Sample input files (input.dat), readme.txt, report.doc, and a checker to verify your program results can be found at the NTU COOL course link below:

https://cool.ntu.edu.tw/courses/285/assignments/1858