I L L I N O I S

VLSI CAD: LAYOUT:
Programming Assignment 2: A Router

Rob A. Rutenbar
University of Illinois at Champaign-Urbana

In lecture, we described how to use the maze routing method to route the wires in a large ASIC, using a 3-dimensional stack of routing grids. In this programming assignment, you get to build a version of a *real* maze-router and run on it some synthetic benchmarks (to test individual routing features) and on some real industrial benchmarks. The assignment has two parts:

- **Required**: Build a **core router.** This will handle 2-point nets, non-unit costs in the routing grid, bend penalties, and 2 separate routing layers. You route the nets in the order they appear in the input file. There are two input files: one describes the geometry of the routing grids (.grid file); the other describes the netlist of wires to connect (.nl file). We will give you a series of increasingly complex benchmarks that test specific features of the router.
- **Optional Extra Credit:** We will give you some bigger industrial benchmarks to play with. You can try your final router on these, and see what happens. Even if your router does not implement all the functionality of the most complex router, you can probably route a lot of nets in these designs, and acquire some additional points.

You can write this program in any language you choose. We will provide inputs as a pair of simple textfiles: a netlist file (which tells you what nets to route, and where their pins are) and grid file (while tells you what the routing grid looks like, where the obstacles and non-uniform cells costs are). You will output your final routed result as a simple textfile in a fixed format. You upload your results to the Coursera site and our auto-grader will score your results.

[100 points] Core Router

The nice thing about maze routing is that the “serious core” of the algorithm is not very complex, when implemented as software. And, one can add features in an incremental way, by starting with a simpler router, and adding code to add these features. So, we suggest you follow a strategy where you first build and test the most basic features, before you add more complicated features. And, we have skewed the points in the project so that you get a lot of points quickly by implementing these basic features first.

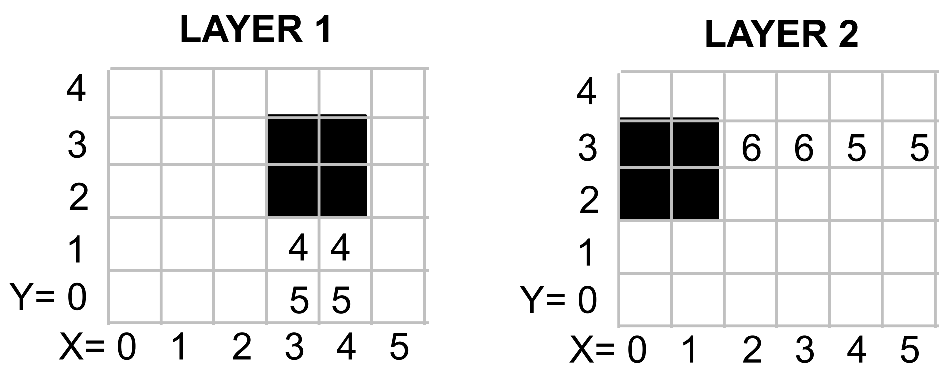
Core Router Logistics

Input file formats: You need to know two things to do this router. First, you need to know what the routing grids look like, so you can search for paths using the maze routing method from lecture. Second, you need to know what the netlist is – what wires we want to route, where their pins are. So, we use simple text formats to specify each placement problem. Your router will read two input files: the **netlist** file (which wires to route, where are their pins) and the **grid** file (describes the routing grids).

The **grid** file specifies the physical grid on which your maze router will search. It has this format:

- **First line:** four integers: **X grid size** (columns), **Y grid size** (rows), the **Bend penalty** (for use in maze search) and the **Via penalty** (also for use in maze search)
- **The next X * Y integers in the file:** specify the costs associated with each grid cell in **LAYER1** of the routing. Most cells will be specified with a 1: these are free to route on, and they have unit cost. If a cell has a non-uniform cost, it will appear as a **positive** number. If the cell is blocked (you cannot use it to route), it will appear as a **negative** number, specifically a -1.
- **The final X * Y integers in the file:** specify the costs associated with each grid in the **LAYER2** of the routing, using the same rules as LAYER1.
- **Grid cell order:** We specify the grids in coordinate order: x coordinates from 0 to **X - 1**; y coordinates from 0 to **Y - 1**.

Here is a small example with the grids drawn in detail, and the associated input grid file shown. A white cell is available to route and has cost=1. A black cell is an obstacle, unavailable to route. A small positive integer shows a free cell with non-uniform routing cost.



Here is a detailed text file input for these grids. Note: the comments (marked by // …) are for illustration purpose, we do not put any comments in your input grid file.

```
//EXAMPLE GRID FILE FORMAT FOR ABOVE GRID DIAGRAM
6 5  5 10     // X gridsize  Y gridsize  BendPenalty  ViaPenalty
1 1 1 5 5 1   // 1st row of LAYER1, where y=0
1 1 1 4 4 1   // 2nd row of LAYER1, where y=1
1 1 1 -1 -1 1 // 3rd row of LAYER1, where y=2
1 1 1 -1 -1 1 // 4th row of LAYER1, where y=3
1 1 1 1 1 1   // 5th row of LAYER1, where y=4
1 1 1 1 1 1   // 1st row of LAYER2, where y=0
1 1 1 1 1 1   // 2nd row of LAYER2, where y=1
-1 -1 1 1 1 1 // 3rd row of LAYER2, where y=2
-1 -1 6 6 5 5 // 4th row of LAYER2, where y=3
1 1 1 1 1 1   // 5th row of LAYER2, where y=4
```

The netlist file specifies the nets you need to route. Here are the constraints on our nets, for this programming assignment:

- All have just 2 pins. So, these are "2 point" nets. This simplifies the programming and the grading.
- We specify a **NetID** for each net, which is just an integer. We specify the nets in **NetID** order in this input file: 1, 2, 3.. etc
- We specify an (**X,Y**) coordinate for each pin on each net.
- We specific a layer **L**=1 or 2, for each pin on each net.
- You route the nets **in NetID order**, i.e., in the order we specify the nets in this input file.

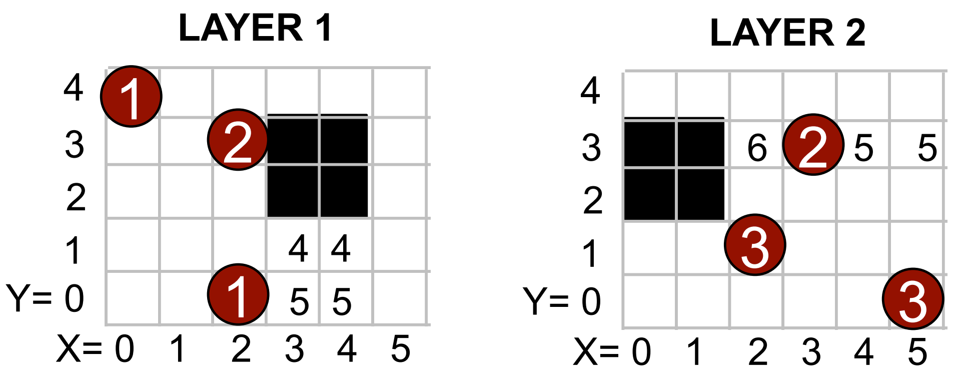
The file format itself is simple:

- **The first line:** one integer: NetNumber, which is how many nets in this problem.
- **The next NetNumber lines:** each specifies one net to route.
- **To specify each net:** Each net is specified with five integers:

  NetID LayerPin1 Xpin1 Ypin1 LayerPin2 Xpin1 Ypin

  So, we tell you the name of the net (NetID) and for each pin (1, 2) the layer, and the (X,Y) coordinates.

As an example, here is the same set of grids, with three nets specified. We have circles with numbers 1, 2, 3 to identify the pins that specify each net.



Here is the netlist input file that corresponds to these nets to be routed. Again, the comments are for explanation only; we do not add comments to the actual file:

```
//EXAMPLE NETLIST FILE FORMAT FOR ABOVE GRID DIAGRAM
3       // NetNumber = 3 nets to be routed
1  1 0 4  1 2 0   // Net 1.  Pin1 Layer1 (0,4);  Pin2 Layer1 (2,0)
2  1 2 3  2 3 3   // Net 2.  Pin1 Layer1 (2,3);  Pin2 Layer2 (3,3)
3  2 2 1  2 5 0   // Net 3.  Pin1 Layer2 (2,1);  Pin2 Layer2 (5,0)
```

Output file format: Your router will find a path for each net, using the cells in the grids and maybe some vias. Or, you might not be able to route a net. Your output tells us what happened with each net you tried to route, and what path it took.

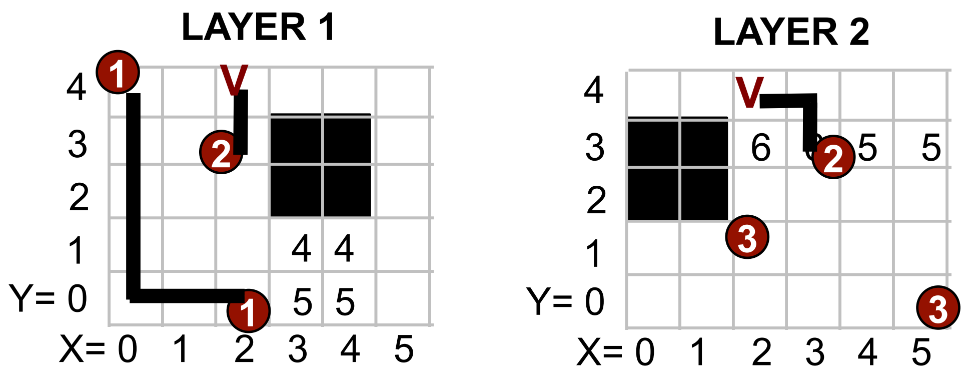
This is also a simple text file, with the following format:

- **First line:** One integer, **NetNumber**, how many nets are in this benchmark. This is just the same as in the input netlist file.
- **For each net in order 1, 2, … NetNumber:** describe the path each net took.
  - **First line: NetID** (an integer)
  - **Next lines:** Describe each cell in the routing path, one cell per line of this file. Each line is 3 integers: **LayerInfo Xcoord Ycoord**.
  - **LayerInfo:** is 0, 1, 2 or 3. If your cell is on Layer1, this is 1; if your cell on Layer2, this is 2. If this is a via, put a 3. To signal the end of this list, put a 0 on the line with nothing else.

Note that you indicate a net that you could not route by just have a single line after the NetID with a 0.

Here is our small example again, now with a real path drawn for each. Note the following:

- **Net 1:** is routed entirely on Layer1.
- **Net 2:** is routed on both layers, and it has a via at (2,4), indicated in the grid with a "V" on each layer.
- **Net 3:** is unrouted. This is just for illustration purposes.



```
//EXAMPLE OUTPUT FILE FORMAT FOR ABOVE GRID DIAGRAM
3       // NetNumber = 3 nets we have tried to route
1       // Net 1 starts here
1 0 4   // First cell on Net1 path is Layer1 at (0,4)
1 0 3
1 0 2
1 0 1
1 0 0
1 1 0
1 2 0   // Last cell on Net1 path is Layer1 at (2,0)
0       // Net1 end
2       // Net 2 starts here
1 2 3   // First cell on Net2 path is Layer1 at (2,3)
1 2 4
3 2 4   // Net2 has a via at (2,4)
2 2 4   // Net2 starts again at (2,4) thru this via on Layer2
2 3 4
2 3 3   // Last cell on Net2
0       // Net2 end
3       // Net3 starts (remember, it was unrouted)
0       // Net3 end
```

The basic idea is you just list **every cell on the path, in order,** from the first pin listed in the netlist file, to the second pin listed in the netlist file. More precisely, we expect you to start routing with the first pin in the netlist file as the source, and to route to the second pin as the target. **List the path in order, from source cell to target cell.**

A few subtle points here, to be clear about:

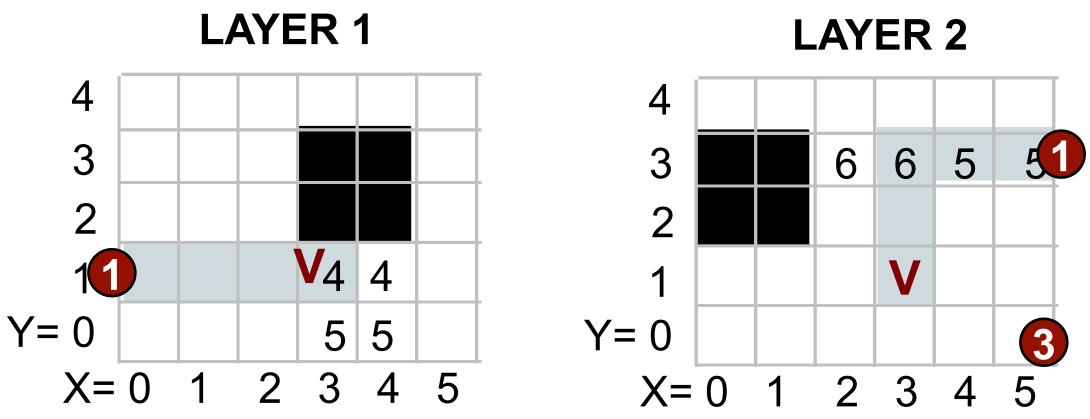
- Make sure you list every cell on the path.
- List the starting source cell as the first cell on the path.
- If there is a via at location (X,Y), you will create **3 lines** in the output file: you got to cell (X,Y) on one layer, so you list this cell on this layer; you put a via at (X,Y), so you list this via; you expanded to the other layer at cell (X,Y), so you list this cell at this other layer, as well. This is very important for the visualization tool and the auto-grader to work properly.

- List the ending target cell as the last cell on your path.
- If you fail to route a path, you still have to list the net. But, you just put the "0" marker on a line, to indicate there are no cells on the path.

Evaluating router results: We have three basic criteria for judging your router's results:

- **Completion**: Did you correctly route the net, yes or no?
- **Cost**: What was the total cost of the path you used to route this net?
- **Shape**: Did your router do the right thing, and correctly find a route that has the expected shape (such as avoiding obstacles, no zig-zag route when bending cost is high, etc.)?

For some benchmarks, we mostly care about the *completions*. For others, we look at the *cost* as well, since we want to know what your router did, with a bit more precision. It is worth noting that the calculation for cost means we add up the cost of all the cells and vias and bends on your routing path. It is useful to look at a concrete example of how this cost calculation works:



The diagram above shows the path (shaded) of Net1, which starts on Layer1 and ends on Layer2, and inserts one via at (4,1). The cost of this path would be computed as follows, cell by cell:

```
Starting on Layer1, from cell (0,1)
1 + 1 + 1 + 4          // 3 unit cost cells, and 1 non-unit cells
+ 10                   // the Via penalty from our Grid input file
+ 1 + 1 + 6            // cell costs on Layer2, going vertically
+ 5                    // Bend penalty to turn from North to East
+ 5 + 5                // final cells on horizontal path to target
----------------------------------------------------------------
40 = Total path cost
```

Building the Core Router: Suggested Strategy

As we mentioned in the introduction, the nice thing about maze routers is you can build a "lot of" router with "a very little" code. So, we suggest you build the router incrementally, by starting with basic functionality, and then add more features. Our benchmarks are organized this way too. You can get a lot of points with a very basic, very simple router. As you add features, you can try the more complex benchmarks, and get more points. We suggest you build the router features in this order:

- **One Layer Unit-cost Router:** this is the simplest router. You expand the source cell until you reach the target cell. Every cell has unit cost. There are no bend penalties. You route on just one layer. With this router, you can run our first benchmark, which has all of its nets on Layer1. So, you read in both grids (for Layer1 and Layer2) but you can ignore Layer2 entirely.
- **One Layer Router with Bend Penalty and Non-Unit Cost:** Starting from the previous router, add bend penalties, and non-unit costs from grid cells. Mechanics to deal with these two features are very similar: you have to add something "different" when you expand a non-unit cell, and you must pay attention to expansions that change directions, and add something else (the bend penalty). To keep things simple, you can stop when you reach the target cell (i.e., ignore other problems with the inconsistent cost function). With this router, you can run our second benchmark, which also has all its nets on Layer1.
- **"No Vias" 2-Layer Router:** Starting from the previous router, add a second routing layer. This router is "No Vias" because we never give you nets that have pins on two different layers. So, you will get some nets that can be routed entirely on Layer1, and some nets that can be routed entirely on Layer2. You do **not** need to implement the vias to route this benchmark. This really means just a few extra features. (1) When you add a cell to the heap, you need to remember what layer it was on. (2) When you print out the path for each wire, remember to print the right layer for each cell.
- **Real 2-Layer Router:** Starting from the previous router, add the vias, so you can change layers as needed in this router. This means just a few extra features. (1) When you expand a cell, as well as north/south/east/west expansion directions, you have to try to expand it to the other layer, i.e., up from layer 1 to 2, or down from layer 2 to 1. (2) When you expand through a via and change layers, you must remember to add in the via cost. (3) When you print out the path for each wire, remember to print not only the right layer for each cell, but also print the vias as well.

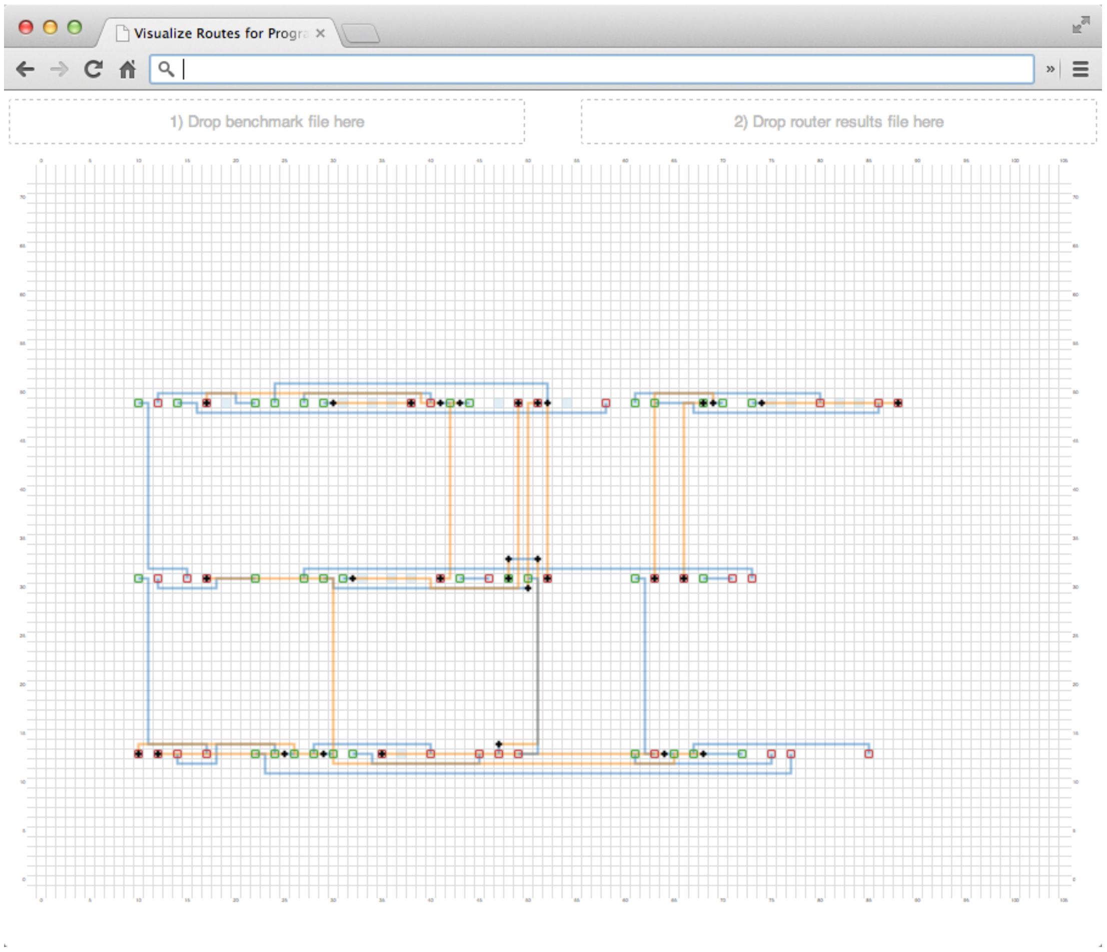
Building the Core Router: Code for Heap

To get you started with the placer problem, we gave you several options for the linear matrix solver. You don't need anything this complicated for the router. But, you do need a **heap**. It turns out heaps are very easy to implement, and widely available. So, we will tell you a few different ways to do heaps in a few common languages, to get you started

- Heap in C++ - Use `priority_queue` from the STL
- Heap in JAVA – Use `PriorityQueue` from java.util package
- Heap in Python – Use `heapq` from the Python Standard Library

Building the Core Router: Visualization

Nobody builds layout tools without being able to *see the results*. This was true for our placer in Program Assignment 1, and it is still true for our router in this programming assignment. Since this is an essential part of this effort, we have *again* provided some very nice visualization tools for this assignment. You can take your textfile output, and visualize the results immediately in your browser as shown in the image below. The web page can be found in https://spark-public.s3.amazonaws.com/vlsicad/javascript_tools/router.html


Visualize Routes for Progr
1) Drop benchmark file here
2) Drop router results file here

Because the visualization tool has quite a few features, and different colors and symbols, we have created a video tutorial to explain it. You can find the video in the tools section on the Video Lectures page at
https://www.youtube.com/watch?v=4m0V6fpwROA

The visualization uses the d3.js library to visualize the results. Some of the larger benchmarks might take some time to draw but we have tested this with the largest benchmark (~1000 x 1000 grid and 1000 routes) and it does work.

Do this:

- **Build** the core router. You read our netlist file and grid file, you write out our output path file format.

- **Run** our benchmarks. We will provide you with 5 separate benchmarks, of increasing router complexity. If you follow the suggested implementation strategy, each new feature will let you run the next set of benchmarks

- **Upload** your results to our Coursera website, and our auto-grader will tell you how you did.

Core Router Grading Details

There are 5 benchmarks. Total assignment is worth 100 points.

- **[25 points] 1-Layer 2-Point Unit-Cost Routing:** we have a simple synthetic benchmark with 20 nets with simple routing patterns, all in Layer 1. Straight lines, simple bends, etc. You get 1 point for being able to route each net. You get 5 points for just having the file format correct and readable by our autograder.
- **[25 points] 1-Layer 2-Point Bend-Penalty Non-Unit-Cost Routing:** we have a simple synthetic benchmark with 20 nets in a grid with non-uniform costs. Again, we use simple routing patterns, all in Layer 1. Now, we look not only at your ability to connect each net, but the cost of each of your nets. We do this by adding up the cost of all the cells on your net: unit cost cells, non-unit-cost cells. You get 0.5 point for being able to route each net. You get 0.5 points for having the net have the shape/path we expect for a properly functioning router (or smaller fractions if your net completes, but has a strange shape). You get 5 points for just having the file format correct and readable by our autograder.
- **[20 points] "No Via" 2-Layer 2-Point:** This synthetic test has 16 nets, 8 nets in Layer1, 8 nets in Layer2, and you do not need to implement the vias. You just need to be able to read a netlist with nets on 2 different layers, and route them on these isolated layers. 1 point per net. 4 points for having the file format correct and readable by our autograder.

- **[15 points] Real 2-Layer 2-Point Router:** This synthetic test has 15 nets and several of these nets have pins that require you to cross the layers. If you can't deal with the vias, you can't route this benchmark. 1 point per net. (By the time you get to this benchmark, we assume you can do the file formats correctly.)
- **[15 points] Industrial 2-Layer Design:** This is a semi-real benchmark from our placer programming assignment. This is the **fract** benchmark from that assignment, placed in a simple grid, assuming a simple standard cell library. We place the standard cells in rows with some fixed, extra spacing for the wires. We model pins as being arbitrarily located inside the cells, all on Layer1. You have 2 routing layers to use to route these. We have simplified the netlist, because the real benchmark has many nets with more than 2 pins; we have arbitrarily selected a pair of pins from a subset of these multi-pin nets. There are 128 nets in this benchmark. We do not expect you to be able to route all these nets! We do expect you to be able to perform approximately as well as our reference router, which is implementing the maze routing method exactly as we specified in the lecture. Note that for this benchmark, we only care about how many nets you can route. We will not compare your costs with our reference router.

  This industrial benchmark has one special "constraint" you need to be aware of. We put blockages on all the pin locations, for this test case. The idea is simple: if we don't do this, an early net you route might block all the pins you need to use to start/end some future nets. So, the way you deal with is simple: **when you route a new net, *unblock* the source and target pins as the first step**.

  Also, we recommend you take a look at some of the ideas for the Extra Credit, since you might want to try those on this test case as well, if you are feeling ambitious.

In conclusion, for the first four benchmarks, your score basically consists of four parts:

- Points for valid format. This only exists in simple benchmarks to make sure you have a correct format for the grader.
- Points for successfully routed the nets and having reasonable costs. If the cost of a net is too high comparing to our reference router, it means your path has a weird shape or taking detour. You receive points according to the following:
  - You get 1/2 of the points for successfully routed a net.
  - You get 1/2 of the points if the cost is within 2X of reference cost.
  - You get 1/4 of the points if the cost is within 4X of reference cost.
  - You get 1/8 of the points if the cost is within 8X of reference cost.
  - Otherwise, you will not receive points for cost being too high.

- Points for having expected shapes. If you implement the router correctly, it should be able to find valid routes that have the expected shapes.

For the semi-real industrial benchmark, it will be scored with respect to the **completion rate**: how many nets did you route? The number of nets routed by your router is expected to be close to a fraction of the number of nets routed by our reference router. You get full points for a benchmark if you successfully routed this many nets.

Please refer to the programming assignment page for details about the grader output.

Core Router Submission and Deadlines:

- You can submit as many times as you like.
- Deadline is the usual "two Tuesday's after assignment appears"**.**

Acknowledgements:

- The industrial benchmark set is again from the Microelectronics Center of North Carolina (MCNC) from the early 1990s. They are referred to in publications from that era as "**the MCNC benchmarks**". We have made some modifications to these to make them suitable for our use in this MOOC.
- Thanks for TA **Zigang (Ivan) Xiao** for the reference Core Router and the auto-graders.
- Thanks to TA **Nicholas Chen** for the visualization drag-and-drop function, and the integration of our auto-graders.
- Thanks to TA **Tsung-Wei Huang** for rebuilding the machine problem to the docker format.