National Taiwan University
Department of Electrical Engineering
Algorithms, Fall 2025

Handout #32
November 19, 2025
Chien-Mo Li & Chin-Lin Wu & Cheng-Yu Chiang

Programming Assignment #3 (due at 12:00 noon, December 21, 2025)

Modified from the 2024 ISPD Global Routing Contest

1. Problem: Global Routing

Imagine you are designing a complex computer chip, like a CPU or GPU. This chip has millions of components (like transistors) that must be connected by microscopic wires. The process of finding the paths for all these wires is called **routing**.

In this assignment, you will implement a global router. You can think of this as planning the general "highways" for wires on a 3D map.

- The Resources: Your resources are represented as a 3D grid made of multiple 2D layers, stacked on top of each other. The map is divided into rectangular "districts" called GCells (Grid Cells).
- The Netlist: You are given a list of nets. Each net needs to connect multiple specific points (called pins) on the map.
- The Challenge: The "roads" (GCells) have a limited amount of space, or capacity. You cannot route too many wires through the same GCell. A traffic jam is called congestion or overflow.

Your goal is to find a path for every net that connects its pins. Your solution will be judged on two primary criteria:

1. Minimize Congestion: You must avoid "overflow" as much as possible. Excessive congestion indicates that the routing demand exceeds the available capacity, which can lead to design rule violations and routing failure.
2. Minimize Wirelength: After minimizing congestion, you should try to make the total length of all your wires as short as possible.

This problem models a real-world challenge in chip design. Finding good routes is essential for a chip to be fast, efficient, and manufacturable. Because the full routing problem is highly complex, **we simplify it in this assignment by restricting the map to two layers and limiting each net to two pins.**

2. The Routing Environment

You are given a set of components that define the "map" for routing.

- Grid: A 2-layer grid composed of $2 \times xSize \times ySize$ GCells (2 layers, $xSize$ columns, $ySize$ rows).
  - A GCell g at row i, column j, and layer $l \in \{0, 1\}$ is denoted $g_{l,j,i}$.
- GCell Properties:
  - Capacity (C_g): The maximum demand (an integer) that GCell g can support.
  - Horizontal Distance (W_j): The physical length of a horizontal segment connecting column j and column $j+1$ (i.e., between $g_{l,j,i}$ and $g_{l,j+1,i}$). There are $xSize - 1$ such distances, $W_0, \ldots, W_{xSize-2}$.
  - Vertical Distance (H_i): The physical length of a vertical segment connecting row i and row $i+1$ (i.e., between $g_{l,j,i}$ and $g_{l,j,i+1}$). There are $ySize - 1$ such distances, $H_0, \ldots, H_{ySize-2}$.
- Nets: A set of nets $N = \{n_1, n_2, \ldots, n_{|N|}\}$.
  - Each net n_i consists of exactly two pins $(p_{i,1}, p_{i,2})$ located in specific GCells.
- Via Wirelength Cost:
  - WL_{via}: A constant integer cost (`UnitViaCost`) associated with using a via. This cost is added to the total wirelength.

3. Routing Rules and Constraints

To find a valid solution, you must connect all nets while adhering to the following rules. We can divide these into hard constraints (which you must follow to have a valid solution) and soft constraints (which you must optimize to get a good cost).

3.1. Inputs and Outputs

Your program will read the routing resource and net information from input files (`.cap`, `.net`) and write your routing solution to an output file (`.route`). The routing resource gives $2 \times xSize \times ySize$ grids and their costs and capacities. The net information gives $|N|$ nets and their pin location. Assume $xSize, ySize \in \mathbb{Z}_{>0}$ with $xSize <$ XXX and $ySize <$ XXX, and $N \in \mathbb{Z}_{>0}$ with $|N| <$ XXX. Detailed formats are described in Section 5.

3.2. Constraints (Pass/Fail Requirements)

- Pin Connectivity: A routing path P_i for a net n_i is only valid if it forms a continuous route that starts at the GCell containing $p_{i,1}$ and ends at the GCell containing $p_{i,2}$.
- Valid Moves: A path is built from a sequence of GCells. Each step in the path must be one of the following three valid moves:
  - Horizontal Segment: Connects $g_{l,j,i}$ and $g_{l,j+1,i}$ (adjacent columns, same row, same layer).
  - Vertical Segment: Connects $g_{l,j,i}$ and $g_{l,j,i+1}$ (adjacent rows, same column, same layer).
  - Via: Connects $g_{0,j,i}$ and $g_{1,j,i}$ (same row, same column, adjacent layers).
- Layer Direction: Each layer l has a routing direction (Horizontal "H" or Vertical "V"), specified in the `.cap` file. All wire segments on that layer must follow this direction.
  - If `layerDirection` is "H", only horizontal segments are allowed on that layer.
  - If `layerDirection` is "V", only vertical segments are allowed on that layer.
  - Vias are allowed regardless of layer direction.
- Path Continuity in Output: For each net, the path printed in the `.route` file must be continuous. The "to" coordinate of one line must be identical to the "from" coordinate of the next line in the same net.
- No Zero-Length Segments: Each line in the `.route` file must describe a transition between two *distinct* GCells. Lines where $(l_1, j_1, i_1) = (l_2, j_2, i_2)$ are prohibited and will be rejected by the checker.
- Chip Boundary: All coordinates appearing in the solution must lie within the valid range of layers, rows, and columns as specified in the `.cap` file.

3.3. Objectives (Optimization Goals)

- Congestion: The total "demand" (number of wires) in a GCell g should not exceed its "capacity" C_g. Exceeding this is called overflow, which is the primary cost you must minimize.
- Wirelength: The total physical length of all wire segments and the total cost of all vias should be minimized.

4. Objective and Evaluation

All submissions will be evaluated using a multi-stage process. A solution must pass the initial constraints to be eligible for scoring. The final ranking is determined by a strict two-tier hierarchy. The individual score per test case is determined by the correctness of the output result as well as the file format. (Note that there are more hidden test cases for evaluating your program.)

4.1. Constraints (Pass/Fail)

A submission will be graded only if it meets the following criteria:

- Validity: The output `.route` file must be valid.
  - All nets must be routed from their source pin to their target pin.
  - All routing paths must be continuous.
  - All segments and vias must adhere to the Routing Rules (e.g., layer-preferred direction, valid via placement, chip boundary constraint).
  - A solution that fails to route all nets or violates any rule is considered invalid and will receive a score of zero.

- Runtime Limit: The program must complete execution within 60 minutes on EDA Union Server.

4.2. Scoring (Hierarchical)

The evaluation considers total overflow, wirelength, via usage, and runtime to comprehensively assess the quality of each solution. All valid solutions that are completed within the time limit will be ranked. The ranking follows two strict stages.

Stage 1: Primary Objective (Total Overflow O_{total})

The primary objective is to minimize total overflow.

- Demand (D_g): Congestion and demand are defined only on GCells. For each GCell g, D_g is the number of distinct nets that are incident on g. A net is counted as incident on g if any of the following is true:
  - one of its pins is located in g,
  - at least one of its wire segments lies on g, including both endpoints
  - at least one via locates on g.

  Each net contributes either 0 or 1 to D_g; passing through g multiple times does not increase D_g further. Vias do not contribute an additional demand. In other words, a via may cause a net to be incident on g, but it does not add an extra unit of demand beyond the net's presence in that GCell.
- Overflow (O_g): The overflow of GCell g is calculated as:

$$O_g = \max(0, D_g - C_g)$$

The total overflow for the solution is:

$$O_{\text{total}} = \sum_{\text{all } g} O_g$$

Solutions are ranked first by their O_{total}.
If two solutions have the **same** O_{total}, their relative rank is determined by the secondary scoring stage.

Stage 2: Secondary Objective (Total Cost $WLRT$)

Solutions are ranked by a Total Cost $WLRT$ (lower is better), which is the total wirelength weighted by a runtime factor.

1. Total Wirelength (WL_{total}):
   - This is the sum of the physical lengths of all net segments plus a constant wirelength cost WL_{via} for each via used.
   - A horizontal segment connecting $g_{l,j,i}$ and $g_{l,j+1,i}$ (between col j and $j+1$) has a length of W_j.
   - A vertical segment connecting $g_{l,j,i}$ and $g_{l,j,i+1}$ (between row i and $i+1$) has a length of H_i.
   - Let vias(P_k) represents the total number of individual vias used in the path P_k for net k.

- Then, the total wirelength for a net k is calculated as:

$$\mathrm{WL}(P_k) = \left(\sum_{\text{segments } s \in P_k} \text{length}(s) \right) + (\text{vias}(P_k) \times \mathrm{WL}_{\text{via}})$$

- The total wirelength objective is the sum over all nets:

$$\mathrm{WL}_{\text{total}} = \sum_{k \in N} \mathrm{WL}(P_k)$$

2. Runtime Factor: This factor rewards faster solutions.
   - T_{self} = Your program's elapse time.
   - T_{median} = The median elapse time of all valid submissions for that testcase.
   - The factor is calculated as:

$$\text{runtime\_factor} = 0.02 \times \log_2 \left(\frac{T_{\text{self}}}{T_{\text{median}}} \right)$$

   - The factor is bounded to $\pm 10\%$:

$$\text{runtime\_factor\_bounded} = \max(-0.1, \min(0.1, \text{runtime\_factor}))$$

3. Total Cost $WLRT$: This is the ultimate metric for all routable solutions.

$$WLRT = WL_{\text{total}} \times (1.0 + \text{runtime\_factor\_bounded})$$

4.3. Score Composition

Your score will be composed as follows:

- 80% correctness: Whether all nets are correctly connected, the output format is valid, and all hard constraints are satisfied. Solutions may still have overflow. Failing to conform the correctness constraints will automatically result in 0 performance score.
- 20% performance: The quality of your solution among all valid submissions, based on overflow, wirelength, and runtime (as captured by O_{total} and $WLRT$).

Summary of Ranking

Given two valid solutions, Solution A and Solution B:

- Step 1: Compare O_{total}.
  - If $O_{\text{total}}(A) < O_{\text{total}}(B)$, Solution A is better.
  - If $O_{\text{total}}(A) > O_{\text{total}}(B)$, Solution B is better.
- Step 2: *Only if* $O_{\text{total}}(A) == O_{\text{total}}(B)$, compare their Total Cost.
  - If $WLRT(A) < WLRT(B)$, Solution A is better.
  - If $WLRT(A) > WLRT(B)$, Solution B is better.

5. Input / Output Format

5.1. Routing Resource File (`.cap`)

This file specifies the info of the grid.

Dimensions ($2 \times xSize \times ySize$)
2 $xSize$ $ySize$
Via wirelength cost
wlViaCost
Horizontal Distance between GCells ($xSize - 1$ values)
W_0 W_1 ... $W_{xSize-2}$
Vertical Distance between GCells ($ySize - 1$ values)
H_0 H_1 ... $H_{ySize-2}$

Layer info and capacities of all GCells
$C_{g_{l,j,i}}$ represents the capacity of GCell at row i, column j, and layer l
Layer Direction: H/V (Horizontal or Vertical)

Layer 0
layerName_0 layerDirection_0
$C_{g_{0,0,0}}$ $C_{g_{0,1,0}}$ ... $C_{g_{0,xSize-1,0}}$
$C_{g_{0,0,1}}$ ...
⋮
$C_{g_{0,0,ySize-1}}$ ... $C_{g_{0,xSize-1,ySize-1}}$

Layer 1
layerName_1 layerDirection_1
$C_{g_{1,0,0}}$ $C_{g_{1,1,0}}$ ... $C_{g_{1,xSize-1,0}}$
$C_{g_{1,0,1}}$ ...
⋮
$C_{g_{1,0,ySize-1}}$ ... $C_{g_{1,xSize-1,ySize-1}}$

5.2. Net File (`.net`)

This file specifies the nets to be routed. Each net has exactly two pins.

NetName0
(
  (l j i) → Pin 1 GCell coordinates: (layer l, column j, row i)
  (l j i) → Pin 2 GCell coordinates: (layer l, column j, row i)
)
NetName1
(
  (l j i)
  (l j i)
)
...

- NetName: The name of the net (e.g., NetName0).
- (l j i): The GCell coordinates of a pin. The coordinates (l, j, i) represent (layer, column, row).

5.3. Solution File (`.route`)

This file describes the routing path for each net. The path is defined by a sequence of GCell coordinates.

NetName0
(
  l_1 j_1 i_1 l_2 j_2 i_2 → Segment 1 (from GCell 1 to GCell 2)
  l_2 j_2 i_2 l_3 j_3 i_3 → Segment 2 (from GCell 2 to GCell 3)
  l_3 j_3 i_3 l_4 j_4 i_4 → Segment 3 (from GCell 3 to GCell 4)
  ...
)
NetName1
(
  ...
)

Each line inside the parentheses `()` defines one segment or a via of the path, connecting GCell (l_1, j_1, i_1) to GCell (l_2, j_2, i_2).

- (l, j, i) represents (layer, column, row).
- The path must be continuous: the "to" coordinate of one line must be the "from" coordinate of the next line (as shown in the example).
- Each line must describe a movement between two distinct GCells; zero-length segments with identical endpoints are not allowed.

Each line must be one of the following:

- Wire Segment: A connection on a single layer between adjacent GCells.
  - $l_1 == l_2$ (Same layer)
  - And one of the following must be true:
    - Vertical Segment: $j_1 == j_2$ (same column). *(Must match layer direction).*
    - Horizontal Segment: $i_1 == i_2$ (same row). *(Must match layer direction).*
- Via: A connection between layers at the same (row, column) location.
  - $j_1 == j_2$ (Same column)
  - $i_1 == i_2$ (Same row)
  - $|l_1 - l_2| == 1$ (Connects adjacent layers).

5.4. Execution

We will compile and execute your program using commands similar to the following:

```
make clean
make
./bin/router --cap case.cap --net case.net --out case.route
```

Your program should support the above command-line options for specifying the input `.cap` file, the input `.net` file, and the output `.route` file.

6. Example

The sample solution is shown below.

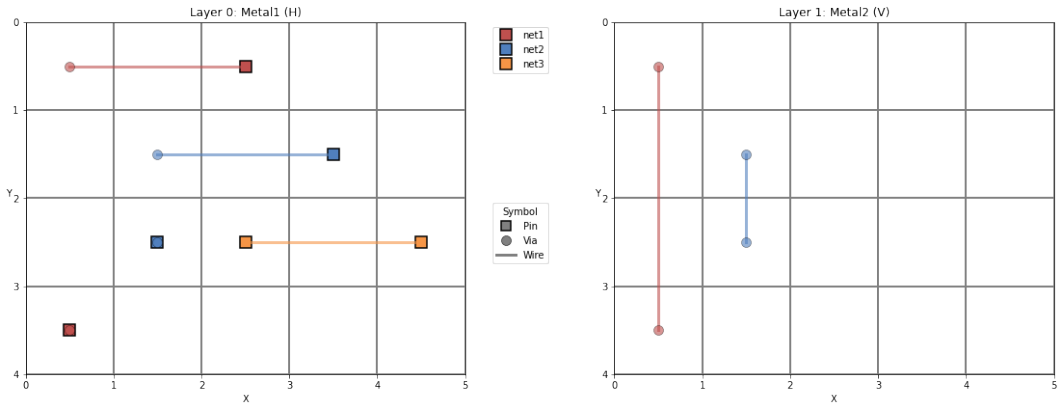

Figure 1: Sample solution

Sample Input (.cap)	Sample Input (.net)	Sample Output (.route)
2 5 4 100 6000 6000 6000 6000 5700 5700 5700 Metal1 H 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 Metal2 V 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	net1 ((0, 2, 0) (0, 0, 3)) net2 ((0, 3, 1) (0, 1, 2)) net3 ((0, 2, 2) (0, 4, 2))	net1 (0 2 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 3 1 0 3 0 0 3) net2 (0 3 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 2 1 1 2 0 1 2) net3 (0 2 2 0 4 2)

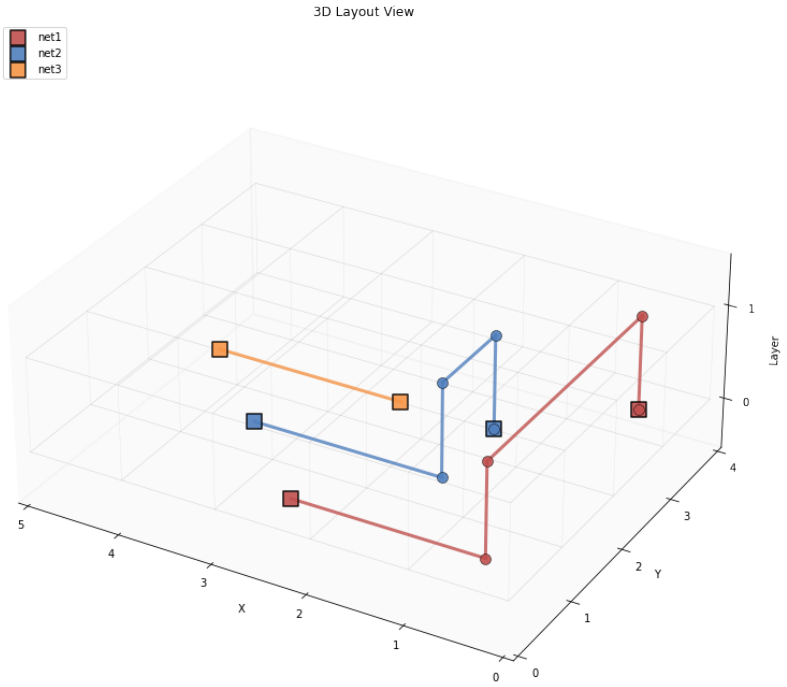

Figure 2: Sample solution (3D)

7. Hint

The routing problem can be modeled as a graph with GCells as nodes. Edge costs are defined to represent wirelength and/or congestion. This graph-based problem can then be solved using Dijkstra's shortest path algorithm. Note that different edge costs yield different results; for example, an exponential cost like $2^{(\text{demand}-\text{capacity})} - 1$ heavily penalizes routing through overfull GCells.

In addition, you may use some strategy to reduce overflow, such as iteratively updating edge costs based on current congestion, applying rip-up-and-reroute on nets passing through severely overfull GCells with history congestion

costs, designing a more refined congestion model (e.g., history-based or look-ahead costs) to better guide the routing, or choosing a better routing order (for example, routing critical or high-demand nets first).

8. Required Files

You need to create a directory named <**student_id**>**_pa3/** (*e.g.* b07901000_pa3/) **(the student ID should start with a lowercase letter)** which must contain the following documents:

- A directory named **src/** containing your source codes (*e.g.* `router.cpp`): only `*.h`, `*.hpp`, `*.c`, `*.cpp` are allowed in src/, and no directories are allowed in src/;
- A directory named **bin/** containing your executable binary named **router**;
- A directory named **doc/** containing your report;
- A makefile named **makefile** that produces an executable binary from your source codes by simply typing "make": the binary should be generated under the directory <student_id>_pa3/bin/;
- A text readme file named **README** describing how to compile and run your program.
- A report named **report.pdf** on the data structures and algorithms used in your program and your findings in this programming assignment. Your report should clearly describe at least the following components in separate, well-labeled sections:
  - The data structures used to store the grids, capacities, nets, and any auxiliary information.
  - The cost functions you design for the graph edges.
  - Your routing strategies and algorithmic flow (e.g., net ordering, path finding).

We will use our own test cases, so you do NOT need to submit the input files. Nevertheless, you should have at least the following items in your *.tgz file.

```
src/<all your source code>
bin/router
doc/report.pdf
makefile
README
```

9. Submission Requirements

1. Your program must be compilable and executable on EDA union servers.
2. The runtime limit for each test case is **60 minutes**.
3. The memory limit for each test case is **30 GB**.
4. Please use the following command to compress your directory into a *.tgz* file:

   ```
   tar zcvf <filename>.tgz <your directory>
   ```

   The submission file should be named as <**student_id**>**_pa3.tgz** (*e.g.* b07901000_pa3.tgz). For example, if your student ID is *b07901000*, then you should use the command below.

   ```
   tar zcvf b07901000_pa3.tgz b07901000_pa3/
   ```

5. Please submit your *.tgz* file to the NTU COOL system before **12pm, December 21, 2025 (Sunday)**.
6. You are required to run the `utilities/checkSubmitPA3.sh` script to check if your *.tgz* submission file is correct. Suppose you are in the same level as the PA3 directory,

   ```
   bash checkSubmitPA3.sh <your submission>
   ```

   For example,

   ```
   bash checkSubmitPA3.sh b07901000_pa3.tgz
   ```

7. Your program must be implemented using the C or C++ programming languages and shall be compatible with a Unix/Linux operating system platform.

10. Reference

[1] C. Y. Lee, "An Algorithm for Path Connection and Its Applications," *IRE Trans. Electron. Comput.*, vol. EC-10, no. 3, pp. 346–365, Sept. 1961.

[2] P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. 4, no. 2, pp. 100–107, 1968.

[3] W.-H. Liu, W.-C. Kao, Y.-L. Li, and K.-Y. Chao, "NCTU-GR 2.0: Multithreaded Collision-Aware Global Routing With Bounded-Length Maze Routing," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 32, no. 5, pp. 709–722, May 2013.

[4] J. Liu, C.-W. Pui, F. Wang, and E. F. Y. Young, "CUGR: Detailed-Routability-Driven 3D Global Routing with Probabilistic Resource Model," in *Proc. 57th ACM/IEEE Design Automation Conf. (DAC)*, San Francisco, CA, USA, 2020, pp. 1–6.

[5] J. Liu and E. F. Y. Young, "EDGE: Efficient DAG-Based Global Routing Engine," in *Proc. 60th ACM/IEEE Design Automation Conf. (DAC)*, San Francisco, CA, USA, 2023, pp. 1–6.