IEE PDA_Lab4 Die-to-Die Global Routing

▶ Update log

Table of Contents

Introduction

Global Routing (GR) is an important step in the IC design flow used to determine the approximate paths for signal nets across the design space. This process mainly focuses on assigning "routing resources" and providing a "coarse routing solution" for subsequent Detailed Routing (DR). Its primary goals include:

- **Path Planning**:
  Determine a global connection path for each signal net *without specifying the exact placement of wires.*

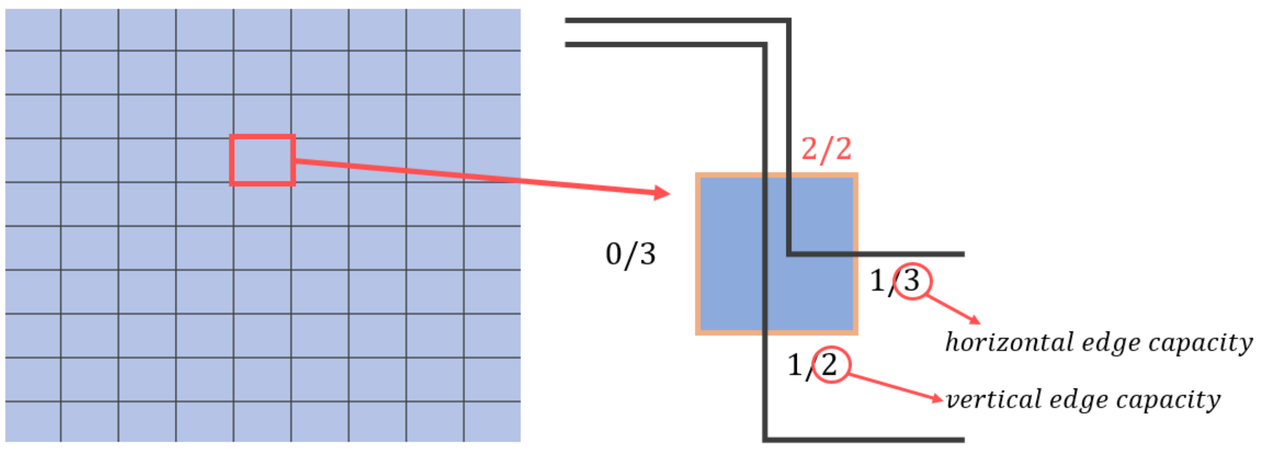
- **Resource Allocation**:
  Ensure that routing resources (such as metal layers and routing tracks) are utilized efficiently across the design to avoid congestion.

- **Feasibility Check**:
  Provide a congestion map to evaluate whether the design is routable and offer optimization suggestions.

- **Performance Optimization**: Minimize routing length and total delay while meeting requirements for timing, power, and reliability.

GCell

Here's a brief introduction to GCell:



Each cell has four surrounding edges, and each of edge has its own capacity, which implies the maximum number of nets that current edge can accept. There also exists GCell that use cell capacity (or even both) to simulate the routing conjestion.
There exists many types of GCell shapes, including rectangular, triangular[1], Hanan-Grid based [2] [3], etc.

A* Search [4]

A* Search is a heuristic search algorithm used to find the shortest path in graphs or grids. It combines the strengths of breadth-first search and heuristic search, making it efficient and widely applicable.

Core Concept

A* uses an evaluation function $\hat{f}(n)$ to select the most optimal node for expansion.

$$\hat{f}(n) = \hat{g}(n) + \hat{h}(n)$$

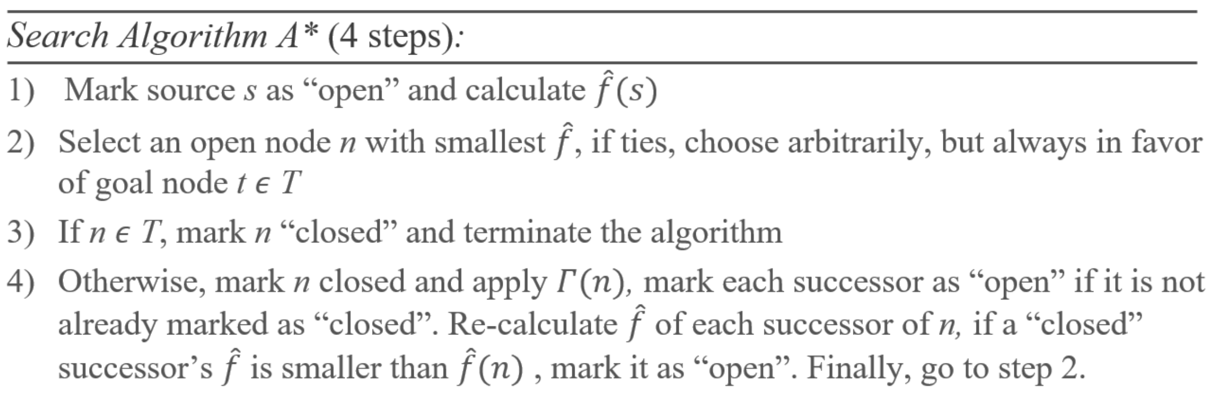
- $\hat{g}(n)$: The actual cost from the start node to the current node n
- $\hat{h}(n)$: The estimated cost from the current node n to the goal node (heuristic function).

Key Features

1. **Optimality**: If the heuristic function $\hat{h}(n)$ is consistent or *admissible*, A* search is guarantees finding the shortest path
2. **Efficiency**: By using $\hat{h}(n)$ to guide the search, A* is faster than pure breadth-first search (i.e. Maze route, or Lee's Algorithm)
3. **Flexibility**: The design of the heuristic function affects both the speed and the outcome of the search

- **Note**: We call an search algorithm *admissible* if it is guaranteed to find an optimal path from source to a preferred goal node

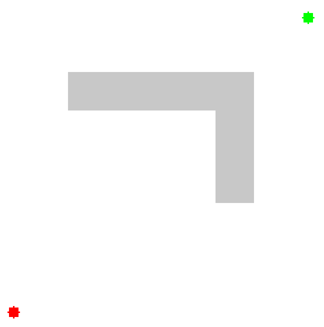
Algorithm



*Search Algorithm A** (4 steps):

1) Mark source s as "open" and calculate $\hat{f}(s)$
2) Select an open node n with smallest $\hat{f}$, if ties, choose arbitrarily, but always in favor of goal node $t \in T$
3) If $n \in T$, mark n "closed" and terminate the algorithm
4) Otherwise, mark n closed and apply $\Gamma(n)$, mark each successor as "open" if it is not already marked as "closed". Re-calculate $\hat{f}$ of each successor of n, if a "closed" successor's $\hat{f}$ is smaller than $\hat{f}(n)$, mark it as "open". Finally, go to step 2.

- T is the set of terminal points of source s
- Define e_{pq} as an arc from node n_p to n_q. If e_{pq} exists, we call n_q a successor of n_p
- $\Gamma(n)$ is a function which returns all successors of node n
- By graph theory, we can prove that it is impossible for a "closed" successor to have a smaller $\hat{f}$ than the current point. Thus, the 4^{th} step can be rewritten as: Otherwise, mark n "closed" and apply $\Gamma(n)$, mark each successor as "open" if it is not already marked as "closed". Finally, go to step 2.



*Source: [Wikipedia (https://en.wikipedia.org/wiki/A*_search_algorithm)](https://en.wikipedia.org/wiki/A*_search_algorithm), licensed under CC BY-SA 3.0.*

Advantages

- Effectively finds the shortest path
- Incorporates heuristic information to reduce the search space

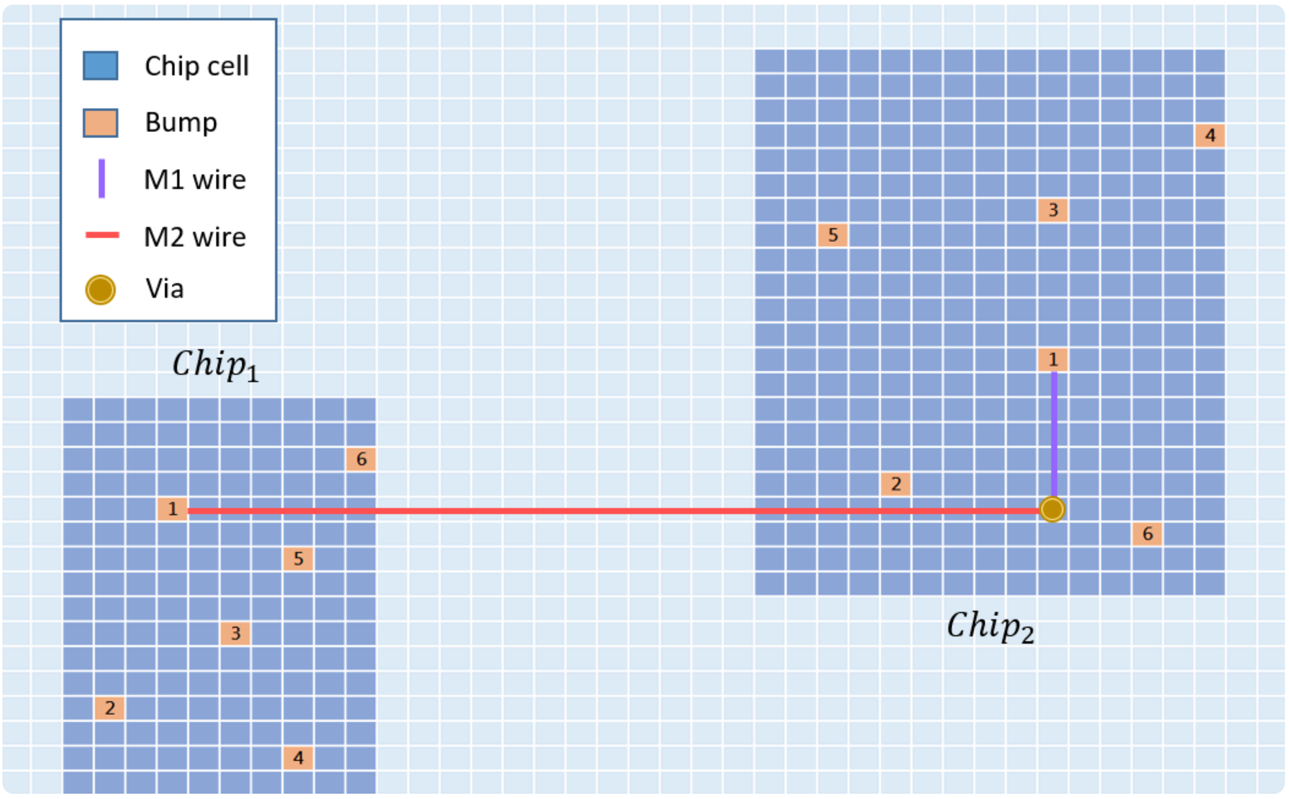
Disadvantages

- Memory consumption can be high when the search space is large
- Poorly designed heuristic functions $\hat{h}(n)$ may lead to inefficiency

- The smaller $\hat{h}(n)$, the more nodes need to be calculated, and will results in low efficiency of the algorithm
- When $\hat{h}(n) = 0$, A* algorithm is same as Dijkstra's algorithm [5]

Problem Formulation

During the GR step, we use GCells to approximate wire allocation. In this lab, we choose a rectangular grid as the shape of the GCells for simplicity.



There will be **two chips** with several bumps, while each bump in the current chip has another corresponding bump in the other chip. In this lab, your task is to complete a die-to-die global routing using the A*-search algorithm[4:1](i.e. to find a minimum-cost path between each pair of bumps).

Your original cost can be calculated by using following formula:

$$oriCost = \alpha \cdot WL_{total} + \beta \cdot OV_{total} + \gamma \cdot cellCost_{total} + \delta \cdot viaCost_{total}$$

where WL is wire length, OV is overflow (i.e. the number of the nets on the edge exceeds its capacity).

Also define the timing factor as below:

$$T = 0.02 * log_2(\frac{T_{yours}}{T_{medium}})$$

$$runtime\_factor = min(0.1, max(-0.1, T))$$

where T_{yours} and T_{medium} is the runtime of yours and medium runtime of all submitted global routers from contestants for current testcase, respectively

$$finCost = (1 + runtime\_factor) \cdot oriCost$$

- The lower your $finCost$, the higher your rank.
- α, β, γ and δ will be given in *.cst with $cellCost$ as .alpha, .beta, .gamma and .delta, respectively
- Define $OV = (netNum - edge_{capacity}) * 0.5 * max(cellCost_{initial})$
  Ex: Consider a placement given by testcase with the largest cell cost 30 and an edge e with capacity 3 passed by 5 net. We get

$$OV_e = (5 - 3) * 0.5 * 30 = 30$$

Routing Constraint

1. In this lab, you can only route in 2 metal layers with restricted direction:
   - M1: Vertical
   - M2: Horizontal
2. Do not route outside the routing area
3. All of the nets should be routed

Input Format

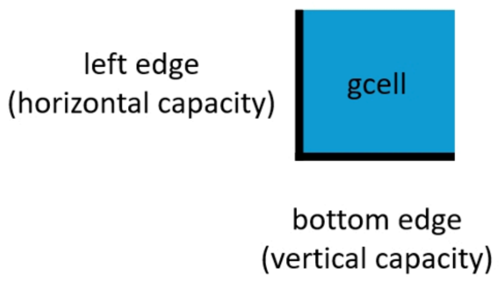
GridMap

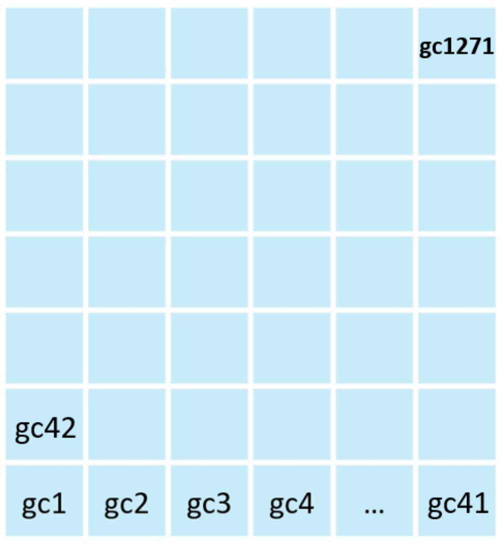
Note that coordinates of chips and bumps are relative to the lower-left of the routing area and corresponding chips, respectively

- Example:
  $routingArea_{corrdinate} = (15, 40), Chip = (20, 0), bump_{coordinate} = (30, 110)$
  then the real chip position is $(15 + 20, 40 + 0) = (35, 40)$,
  real bump position is $(15 + 20 + 30, 40 + 0 + 110) = (65, 150)$

> ▶ gridMap.gmp

GCell





- Each GCell will be given its left and bottom edge capacity
- GCell will be given in raster scan order
  Ex: $GCell_1$ will share the horizontal edge capacity with $GCell_2$ and $GCell_{42}$ will share the vertical edge capacity with $GCell_1$
- TA won't give you the top and right boundary capacity of whole routing area, since it is invalid to route outside routing area

> ▶ GCell.gcl

Cost

- All cell costs will be given as following rules:
  - From left to right
  - From bottom to top

> ▶ cost.cst

Output Format

- Please print the net name as nx, where x is the corresponding bump index.
  - If current net is used to connect $bump_1$, then print $n1$
- When changing layer, remember to place a via
- Every net should start from $M1$, and end at $M1$

- Print the path from $Chip_1$ to $Chip_2$
- The position in *.lg should be
  - real position, not the relative position
  - the lower-left corner of the start/end GCell

Following is the format and an example of *.lg:

```
nx
<layer> <start x> <start y> <end x> <end y>
via
```

> ▶ Example of output.lg

Execution

Your binary file should be executed by following command:

```
$ ./D2DGRter *.gmp *.gcl *.cst *.lg
```

where *.gmp, *.gcl and *.cst are inputs from different testcases, *.lg is your routing result.
Ex:

```
$ ./D2DGRter testcase1.gmp testcase1.gcl testcase1.cst testcase1.lg
```

- **Note**: If parallelization is implemented, only up to **4** threads are allowed. Make sure to limit the number of threads in your program; otherwise, TAs have the right to terminate your program directly

Grading

- Three public cases (60%) and two hidden cases (40%).
  - For each case, correctness (70%) and performance (30%).
  - Performance is based on your $finCost$ for the current case. Only students with a valid routing result will be included in the ranking.

- Cheating or plagiarism will result in a score of 0 for this lab.

- Runtime should less than 600 seconds ; otherwise, you'll fail the case

- For other routing constraints, please refer to Routing Constraint

Submit

- **Duration**: From **2024/11/25 12:00** to **2024/12/16 23:59**
- The top directory should be named as $studentID$
- The directory should contain
  - *.h/ *.hpp (optional)
  - `readme.txt` / `README.md` (optional)
  - *.cpp
  - `Makefile`

  Your directory may looks something like:

```
📁312510224/
├── 📁inc/
│    └── globalRouting.h
├── 📁src/
│    └── globalRouting.cpp
├── main.cpp
├── Makefile
└── README.md
```

- Please tar the file by following command:

```
$ tar cvf studentID.tar studentID
```

- Remember to submit your `studentID.tar` to newE3 (https://portal.nycu.edu.tw/#/login)

Evaluator

Preparation

1. Put all input files and your `.lg` file into a directory
2. Make sure that all of your files have the same names

   For example:

```
📁publicCase/
    └── testcase0.gmp
    └── testcase0.gcl
    └── testcase0.cst
    └── testcase0.lg
```

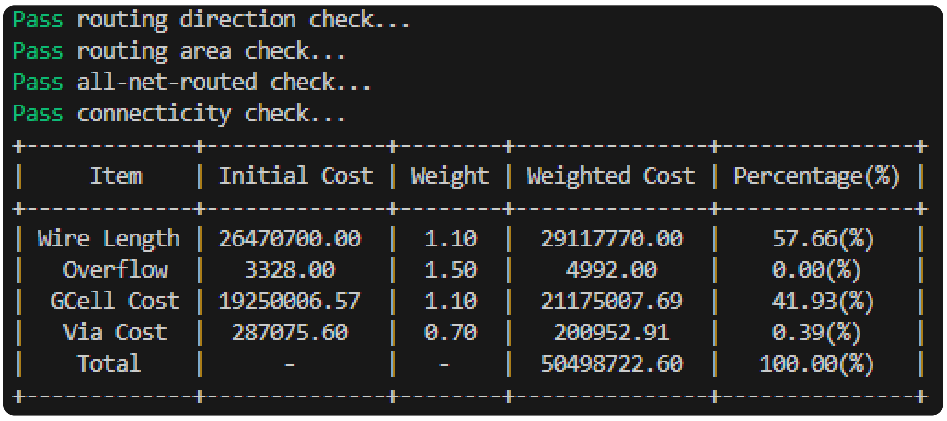
Execution

```
```
$ git clone https://github.com/YubiYubi719/NYCU-PDA-Lab4-Evaluator.git
$ cd NYCU-PDA-Lab4-Evaluator
$ tar xvf Evaluator.tar
$ cd Evaluator
$ chmod 755 Evaluator
$ ./Evaluator <fileDirPath> <testcaseName>

// Ex:
// ./Evaluator ~/publicCase testcase0
```
```

Output Result

If you see a pretty table like this, congratulations!



```
Pass routing direction check...
Pass routing area check...
Pass all-net-routed check...
Pass connecticity check...
+------------+-------------+--------+---------------+---------------+
|    Item    | Initial Cost| Weight | Weighted Cost | Percentage(%) |
+------------+-------------+--------+---------------+---------------+
| Wire Length| 26470700.00 |  1.10  |  29117770.00  |    57.66(%)   |
|  Overflow  |   3328.00   |  1.50  |    4992.00    |    0.00(%)    |
|  GCell Cost| 19250006.57 |  1.10  |  21175007.69  |    41.93(%)   |
|  Via Cost  |  287075.60  |  0.70  |   200952.91   |    0.39(%)    |
|   Total    |      -      |   -    |  50498722.60  |   100.00(%)   |
+------------+-------------+--------+---------------+---------------+
```

Bonus

If you find any bugs, feel free to contact us! Once your observation is verified, you will receive 1 bonus point for this lab.

Q&A

▶ 1. Edge capacity 定義

▶ 2. Evaluator

▶ 3. cost.cst 勘誤

▶ 4. 繞線時是否可以直接跨越其他bump所在之GCell?

▶ 5. GCell cost 與 via cost 之計算方式

▶ 6. 起點、終點之via疑問

▶ 7. 有關Q5之疑問

▶ 8. 是否一定要使用A* search演算法?

▶ 9. 測資之座標是否為整數?

Contacts

Reminder: Please send your questions (or maybe some requests) to both TAs, we'll try our best to answer you within 24 hours. Questions will also be updated in Q&A

- TA1: 林煜睿, yrlin719.ee12@nycu.edu.tw (mailto:yrlin719.ee12@nycu.edu.tw)
- TA2: 陳煥沅, ryan.chen.1104@gmail.com (mailto:ryan.chen.1104@gmail.com)

1. https://ieeexplore.ieee.org/abstract/document/10247899 (https://ieeexplore.ieee.org/abstract/document/10247899) ↩
2. https://en.wikipedia.org/wiki/Hanan_grid (https://en.wikipedia.org/wiki/Hanan_grid) ↩
3. https://ieeexplore.ieee.org/abstract/document/10652868 (https://ieeexplore.ieee.org/abstract/document/10652868) ↩
4. https://ieeexplore.ieee.org/document/4082128 (https://ieeexplore.ieee.org/document/4082128) ↩ ↩
5. https://zh.wikipedia.org/wiki/戴克斯特拉算法 (https://zh.wikipedia.org/wiki/%E6%88%B4%E5%85%8B%E6%96%AF%E7%89%B9%E6%8B%89%E7%AE%97%E6%B3%95) ↩