PA3: Detailed Routing for Photonic Integrated Circuits (PICs)

李品翰
四電機三甲
B11107051

本次作業使用 A* 演算法完成，整體 pseudocode 如下：

Algorithm 1 Overall Code Structure

Input: grid size, propagation loss, crossing loss, bending loss, input nets

Output: route of nets

```
grid ← 2d integer array with all value set to 0                          ▷ 作為地圖
nets ← a empty set of net                                                ▷ 儲存已繞好的線
for input_net in input_nets do
    net ← L_Routing(intput_net)                                          ▷ 無視障礙物，以 L 繞線作為 global routing
    nets ← nets ∪ {net}
    for point in net do grid[point.x][point.y]++ end for
end for
proceed ← true
while proceed == true do                                                 ▷ 存在重新繞過的 net
    proceed ← false
    for net in nets do
        nets ← nets − {net}                                              ▷ 移除舊 net
        for point in net do grid[point.x][point.y]−− end for             ▷ 在地圖上移除舊 net
        new_net ← A*_Routing(grid, net)
        if loss(grid, new_net) < loss(grid, net) then
            nets ← nets ∪ {new_net}                                      ▷ 加入新 net
            for point in new_net do grid[point.x][point.y]++ end for     ▷ 在地圖上加入新 net
            proceed ← true
        else
            nets ← nets ∪ {net}                                          ▷ 復原舊 net
            for point in net do grid[point.x][point.y]++ end for         ▷ 在地圖上復原舊 net
        end if
    end for
end while
```

- `grid` 提供各 net 繞線時，得知某點是否有其他 net 存在，需要大量隨機存取，因此使用 `std::vector`
- `nets` 在 L Routing 後大小維持固定，若找到更佳繞線也可直接覆蓋，沒有特殊需求，因此使用最簡單的 `std::vector`

Algorithm 2 A* Routing Algorithm

Input: grid, input net, propagation loss, crossing loss, bending loss

Output: route of net

```
points ← {first_point(input_net)}                                  ▷ 儲存已探索過的點
while true do
    best_point ← min_total_cost(points)
    lock(best_point)
    if best_point == last_point(input_net) then                    ▷ 找到終點
        return net with points from first_point(input_net) to last_point(input_net)
    end if
    for point in up, down, left, right of best_point do            ▷ 探索周圍點
        if out_of_bound(point) then continue end if
        if point == privious_point(best_point) then continue end if
        cost, estimate_cost = evaluate_cost(point, best_point, grid, last_point(input_net))
        if point ∈ points then
            old_point ← find(points, point)
            if total_cost(point) ≥ total_cost(old_point) then continue end if   ▷ 非更佳路線
            points ← points − {old_point}                          ▷ 新路徑較佳，刪除舊路徑
        end if
        points ← points ∪ {point}
    end for
end while
```

- 探索到終點，準備回傳時，因路徑是由終點一路回推找到起點，需使用到 `push_front`，因此 `net` 使用 `std::deque` 來儲存路徑上的點
- 為了在盡可能多的地方達成 `O(1)` 操作，`points` 被我分成 `id_points` 與 `ordered_points` 兩個 set：
  - `id_points`: 使用 `std::set`，以點座標作為 comparator，避免 set 中出現重複點
  - `ordered_points`: 使用 `std::multiset`，利用 C++ set 排序的特性，依序比較 lock、total cost、estimate cost，此時只要呼叫 `*ordered_points.begin()` 即可以 `O(1)` 時間取得 total cost 最小的點