

Project #2 Report

Introduction - Gordian placer

Option A

Option B

External netlist file

Gordian演算法介紹

Input Format

全局參數

連線 (Net) 格式

元件 (Cell) 格式

分析

How to make the executable

step1 從Github下載 program 並解壓縮

step2 加入數學計算庫

step3 在 Gordian.cpp、Gordian.h、main.cpp 更改"armadillo"的檔案路徑

step4 安裝編譯工具

step5 修改參數

step6 編譯並執行

Result with metrics (Option A)

Case1: netlist_L.txt

case2: netlist_H.txt

case3: netlist_A.txt (External netlist file)

case4: netlist_B.txt (External netlist file)

整理

Result with metrics (Option B)

Case1: netlist_10%5b1%5d.txt

Case2: netlist_30%5b1%5d_class.txt

Case3: netlist_40%5b1%5d.txt

Case4: netlist_A.txt (External netlist file)

整理

Modules analysis

void Gordian::processFunc()

void Gordian::shuffle_()

int Gordian::partition(vector<MatrixInfo_Def> rowinfo_A, mat inputx, mat inputy)

Conclusion

Introduction - Gordian placer

Option A

Author of Gordian.cpp, Gordian.h and main.cpp: Kangli Chu

https://github.com/KangliC/Gordian_Placement

input file x2

netlist_L.txt

netlist_H.txt

Num_Cells=400	Num_Cells=1600
Num_Mcells=394	Num_Mcells=1594
Num_Fcells=6	Num_Fcells=6
Num_Nets=874	Num_Nets=3354
W=60	W=120
H=60	H=120

Option B

<https://github.com/ZhenshenQiu/Gordian-VLSI-Placement>

input file x3

netlist_10%5b1%5d.txt	netlist_30%5b1%5d_class.txt	netlist_40%5b1%5d.txt
Num_Cells=100	Num_Cells=900	Num_Cells=1600
Num_Mcells=94	Num_Mcells=894	Num_Mcells=1594
Num_Fcells=6	Num_Fcells=6	Num_Fcells=6
Num_Nets=234	Num_Nets=1914	Num_Nets=3354
W=30	W=90	W=120
H=30	H=90	H=120

External netlist file

netlist_A.txt

Num_Cells=1000	Num_Cells=2000
Num_Mcells=988	Num_Mcells=1988
Num_Fcells=12	Num_Fcells=12
Num_Nets=2148	Num_Nets=4228
W=90	W=120
H=90	H=120

netlist_B.txt

Gordian演算法介紹

Gordian 是一種經典的 VLSI（超大型積體電路）佈局（Placement）演算法，主要用於將晶片設計中的邏輯單元（如標準細胞、宏模塊）自動放置在晶片平面上，以優化 線長（Wirelength）和 密度（Density），同時避免重疊。它的核心思想是 將佈局問題轉化為數學優化問題，並通過 迭代的二次規劃（Quadratic Placement）來求解。



演算法流程

1. 全域佈局 (Global Placement)

- 將佈局問題建模為一個 **二次線長最小化問題** (Quadratic Wirelength Minimization)，目標是最小化所有互連線的總長度。
- 使用 **矩陣運算 (如稀疏矩陣求解)** 來高效求解線性系統 (例如：解 $Ax=b$)。
- 引入 **密度約束 (Density Constraints)**，確保不會過度集中。

2. 密度控制與迭代優化

- 通過 **加權方法 (Weighting Scheme)** 調整單元的分佈，逐步平衡線長和密度。
- 在每次迭代中，更新單元的位置並調整權重，直到密度均勻。

3. 合法化 (Legalization)

- 將全域佈局的結果 (可能仍有重疊) 調整為無重疊且對齊網格。

4. 詳細佈局 (Detailed Placement)

- 進一步優化單元位置，例如通過局部交換或細微調整來減少線長。

key point

- **矩陣求解**：對於大規模設計 ($>1M$ 單元)，會使用 **稀疏矩陣技術** (如共軛梯度法) 來高效求解線性系統。
- **分治法**：可能結合遞迴分割 (如多層次方法) 來處理超大規模問題。
- **密度罰函數**：通過數學方法 (如拉格朗日乘數) 強制單元均勻分佈。

舉例說明

• 單元與互連：

- 可移動單元：**A**, **B**, **C**, **D** (需佈局到晶片平面上)。
- 固定模塊：**I1** (位置: (0,0)), **I2** (位置: (10,10)) (例如 I/O 端口)。
- 互連關係：
 - **I1** — **A** — **B** — **I2**
 - **C** — **D** — **I2**

• 目標：最小化總線長，同時避免單元過度集中。

1. 建模為二次線長最小化問題

Gordian 將線長最小化問題轉化為 **二次目標函數**。

假設單元位置為 $A=(xA, yA)$, $B=(xB, yB)$, 依此類推，則總線長可表示為：

$$\text{Wirelength} = \sum_{(i,j) \in \text{Nets}} ((x_i - x_j)^2 + (y_i - y_j)^2)$$

忽略固定模塊後，問題簡化為求解以下線性系統 (以 x 方向為例)：

$$\begin{cases} 2x_A - x_B = 0 \\ -x_A + 2x_B - x_C = 0 \\ -x_B + 2x_C - x_D = 0 \\ -x_C + 2x_D = 10 \end{cases}$$

寫成矩陣形式 $\mathbf{M}\mathbf{x}=\mathbf{b}$:

$$\begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} x_A \\ x_B \\ x_C \\ x_D \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 10 \end{bmatrix}$$

2. 求解線性系統

使用 **共轭梯度法 (Conjugate Gradient)** 或其他稀疏矩陣求解器，得到初始解：

$$x_A = 2.5, x_B = 5.0, x_C = 7.5, x_D = 8.75$$

此時單元分佈在 x 軸上，但可能出現密度不均（例如所有單元集中在中心）。

3. Density 控制（加權迭代）

透過 **Density 權重** 強制單元分散：

1. 將晶片平面劃分為網格
(如 2×2 的 Bins)。
2. 計算每個 Bin 的當前密度
(例如 Bin1 有 A、B，Bin2 有 C、D)。
3. 若某 Bin 密度過高，則增加該區域單元的權
重，重新求解線性系統。

4. 合法化與詳細佈局

- **合法化**：將單元對齊到最近的網格點，確保無重疊。
- **詳細佈局**：交換相鄰單元（如 B 和 C）以進一步減少線長。

迭代後結果：

單元位置逐步調整，最終分佈更均勻（例如：A 和 C 向左移動，B 和 D 向右移動）。

⇒ **最終佈局結果**

- 單元座標（近似）：
 - A : (1, 1), B : (6, 4), C : (4, 6), D : (9, 9)
- 總線長：從初始 50（未優化）降低到 30。
- 密度：單元均勻分佈在四個象限中。

Input Format

以 `netlist_L.txt` 為例

```
Num_Cells=400
Num_Mcells=394
Num_Fcells=6
Num_Nets=874
W=60
H=60
```

全局參數

- **Num_Cells=400**：總元件數為 400 個（包括可移動和固定元件）。
- **Num_Mcells=394**：可移動元件（Movable cells）數量為 394 個。
- **Num_Fcells=6**：固定元件（Fixed cells）數量為 6 個。
 - 通常為 I/O pins 或預先固定位置的 macros，例如電源或時鐘單元。
- **Num_Nets=874**：連線（nets）總數為 874 條。
- **W=60、H=60**：布局區域是一個 60x60 的矩形網格，元件將被放置在此區域內。

```
N(1)
C(0)(2,2)(0,1)F(1,1)
C(20)(2,2)(0,-1)M
```

連線（Net）格式

格式：N(id)；後續行：每條連線包含多行 C(...) 描述該 net 連接的元件。

元件（Cell）格式

每個元件行依照以下格式：

```
C(id)(width,height)(x_offset,y_offset)[M|F[(x,y)]]
```

- **C(id)**：元件標識符
 - id：元件的唯一編號（從 0 到 399，總共 400 個元件）
- **(width, height)**：元件的尺寸
 - width：元件的寬度（單位可能是網格單位）、height：元件的高度
 - 觀察檔案，幾乎所有元件的尺寸都是 (2,2)，表示標準化單元（standard cells）
- **(x_offset, y_offset)**：pin 的位置
 - x_offset, y_offset：元件上 pin 相對於中心的偏移量。例：(0,1) 表示 pin 位於元件中心右上方 (x=0, y=1)
 - 常見偏移量包括 (0,1)、(0,-1)、(1,0)、(-1,0)、(-1,-1)、(-1,1)，表示 pin 在元件的不同邊緣或角落
- **[M|F[(x, y)]]**：元件的屬性和位置（若為固定元件）
 - M：表示可移動元件（Movable cell）。
 - F(x, y)：表示固定元件（Fixed cell），並指定其固定位置。
 - x, y：固定元件的座標。例：C(0)(2,2)(0,1)F(1,1) 表元件 0 是固定元件，固定在座標 (1,1)。

分析

簡單連線（2 元件）：大多數連線（例如 N(1) 到 N(360)）連接 2 個元件。

複雜連線（多元件）：從 N(801) 開始，一些連線連接多個元件（例如，N(801) 連接 20 個元件）。

```

N(801)
C(0)(2,2)(-1,-1)F(1,1)
C(21)(2,2)(-1,-1)M
C(42)(2,2)(-1,-1)M
...
C(399)(2,2)(-1,-1)F(59,59)

```

這表示連線 N(801) 是一個 **高扇出 (high fan-out)** 或 **高扇入 (high fan-in)** 的連線，可能代表電源線、接地線或時鐘信號。

pin 位置的規律：

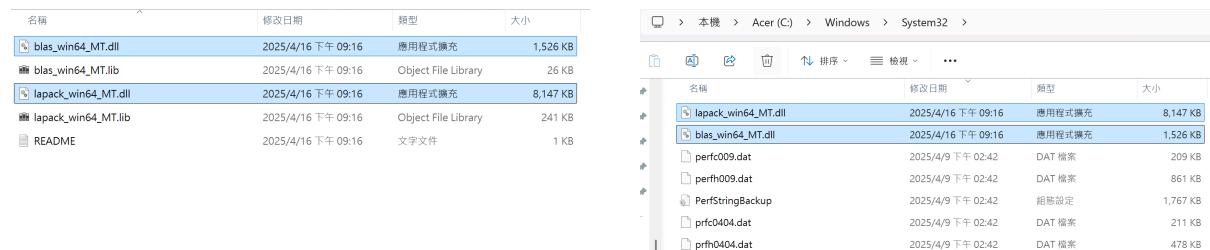
- 早期連線 (N(1) 到 N(360)) 的 pin 位置主要是 (0,1) 和 (0,-1)，表示垂直方向的連接
- 後期連線 (N(376) 到 N(800)) 的 pin 位置變為 (1,0) 和 (-1,0)，表示水平方向的連接
- 從 N(801) 開始，pin 位置多為 (-1,-1) 和 (-1,1)，可能表示更複雜的連接方向 (對角線或多方向)
- 這反映了電路的拓撲結構：早期連線可能是局部連接，後期連線可能是全局信號 (電源或時鐘)

How to make the executable

step1 從Github下載 program 並解壓縮

step2 加入數學計算庫

將兩個 .dll 檔案 (`blas_win64_MT.dll`、`lapack_win64_MT.dll`) 複製到 C:\Windows\System32 資料夾底下



step3 在 `Gordian.cpp`、`Gordian.h`、`main.cpp` 更改 "armadillo" 的檔案路徑

```

Gordian.cpp ➔ main.cpp
Assignment2 (全域範圍)

1 #include <string>
2 #include <cstring>
3 #include <iostream>
4 #include <fstream>
5 #include <vector>
6 #include <list>
7 #include <cstdio>
8 #include <cmath>
9 #include <ctime>
10 #include "EasyBMP_Geometry.h"
11 #include "EasyBMP.h"
12 #include "Gordian.h"
13 #include "C:/Users/user/Documents/課程/EDA/Proj2/Gordian_Placement-master/Assignment2/include/armadillo"
14

```

step4 安裝編譯工具

.vcxproj 專案設定裡指定要使用 **Visual Studio 2013** 的編譯器工具 (v120)

```
<PlatformToolset>v120</PlatformToolset>
```

若沒有使用適合的版本，則會出現下列錯誤訊息

找不到 Visual Studio 2013 的建置工具 (平台工具集 = 'v120')。若要使用 v120 建置工具進行建置，請安裝 Visual Studio 2013 建置工具。或者，您可以選取 [專案] 功能表，或在方案上按一下滑鼠右鍵，然後選取 [重定方案目標]，升級成最新版的 Visual Studio 工具。

解決方法：

✓ 方法 1：安裝 Visual Studio 2013 的建置工具（原始 v120）

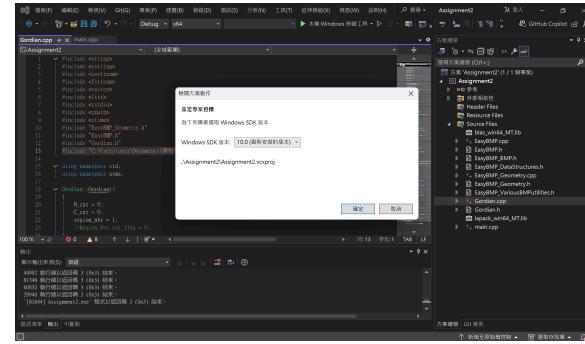
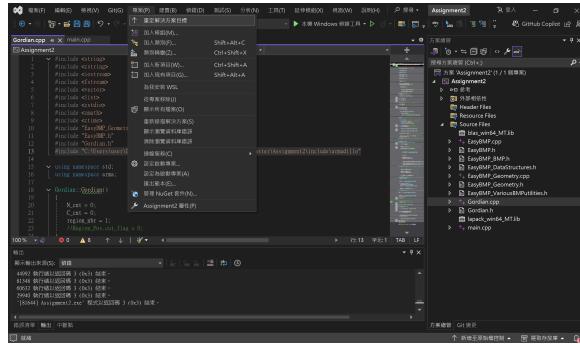
- [微軟官網 Visual Studio Older Downloads](#) 可以找到 VS 2013。

✓ 方法 2：升級平台工具集（最後選擇使用這個方法）

讓專案使用目前安裝的 Visual Studio 工具集版本（比如 v143 for VS 2022）：

✓ 操作步驟：

1. 打開專案（`.sln`）。
2. 點選上方選單的 [專案] → [重定方案目標...]（或右鍵方案 → 選擇）。
3. 在跳出來的視窗中：目標平台版本：選一個電腦已經有安裝的，例如 `v143`。
4. 按下確定，然後重新建置。



✓ 方法 3：手動更改 `.vcxproj`

直接改 `.vcxproj` 也可以，把這行改成現在 Visual Studio 支援的，例如：

```
<PlatformToolset>v143</PlatformToolset>
```

然後再重新打開專案。

step5 修改參數

若希望結果圖上有connection ⇒ 將 702 line in Gordian.cpp 取消註解

若不希望結果圖上有connection ⇒ 將 702 line in Gordian.cpp 取消註解

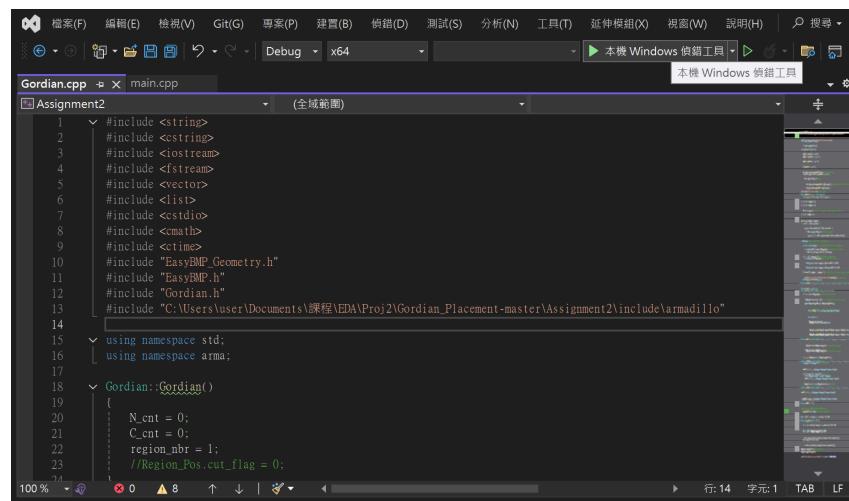
```

for (int i = 0; i < Num_Cells; i++)
{
    plot_a_Cell(Cells.at(i), &image);
}
for (int i = 0; i < Num_Nets; i++)
{
    plot_a_Net(Nets.at(i), &image); //uncomm to have connected plots
}
plot_edges(&image);
char filename[32];
sprintf_s(filename, "Partition_plot_%d.bmp", *fileID);
image.WriteToFile(filename);

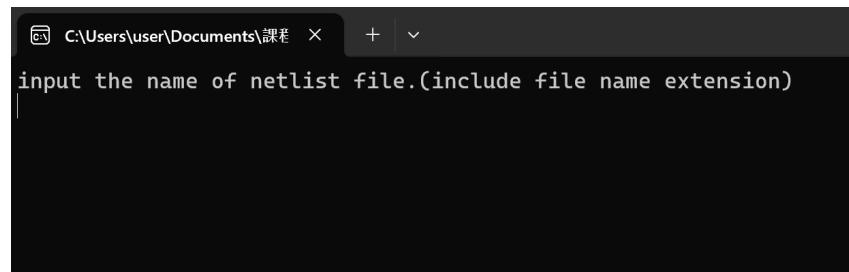
```

step6 編譯並執行

使用本機Windows偵錯工具



在"\x64\Debug"資料夾底下產生執行檔，輸入 input file 名稱即可開始運算

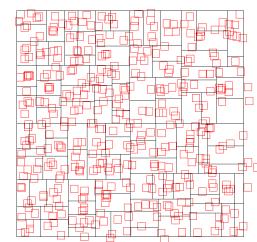
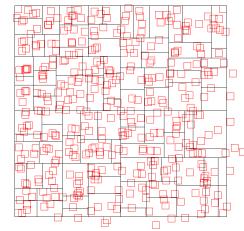
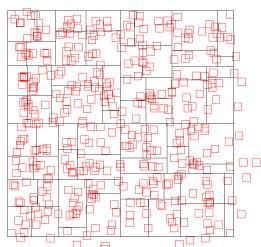
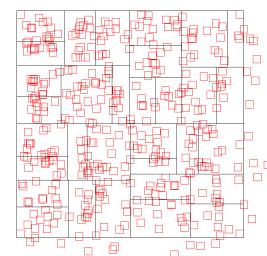
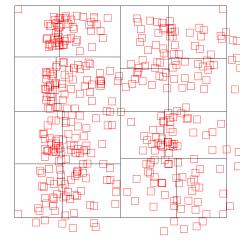
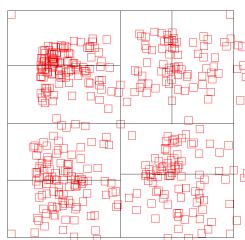
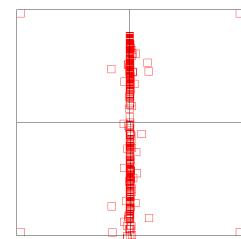
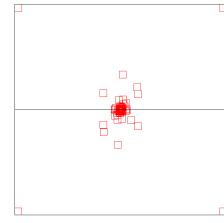
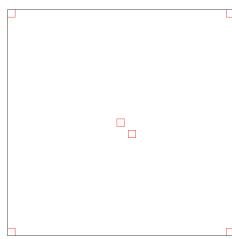


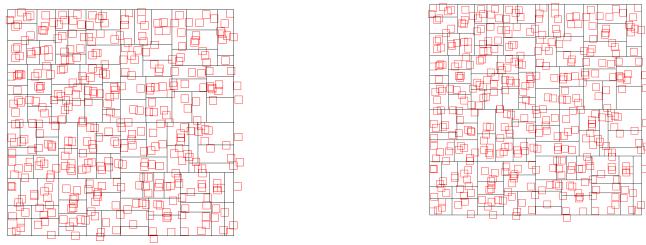
Result with metrics (Option A)

Case1: netlist_L.txt

1. without connection: partition plot 1~11

```
C:\Users\user\Documents\課程 X + ▾  
input the name of netlist file.(include file name extension)  
netlist_L.txt  
partition finish!  
Overall iterations: 10  
Overall wire length is 20539.959(include fixed cells)  
Calculation end!  
Time for Gordian iterations: 1.91
```

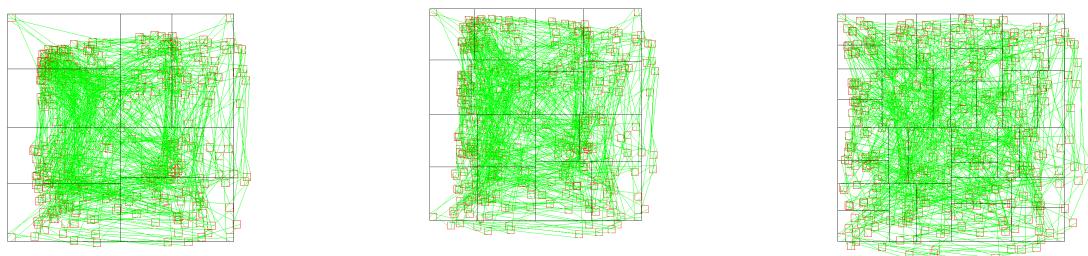
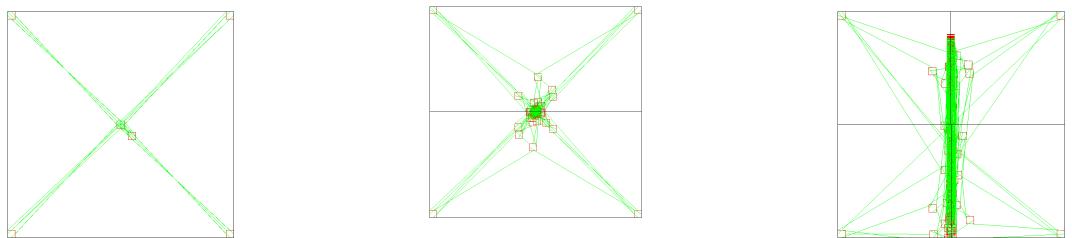


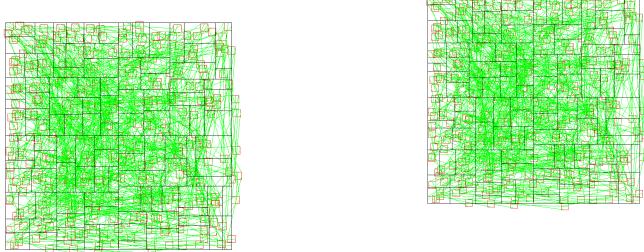
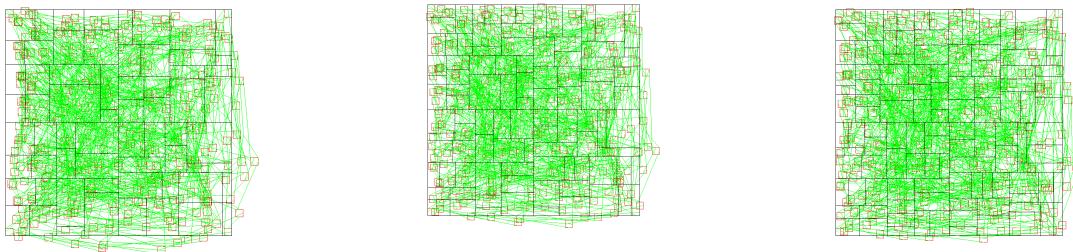


2. with connection: partition plot 1~11

```
C:\Users\user\Documents\課業 x + ^

input the name of netlist file.(include file name extension)
netlist_L.txt
partition finish!
Overall iterations: 10
Overall wire length is 20539.959(include fixed cells)
Calculation end!
Time for Gordian iterations: 1.89
```

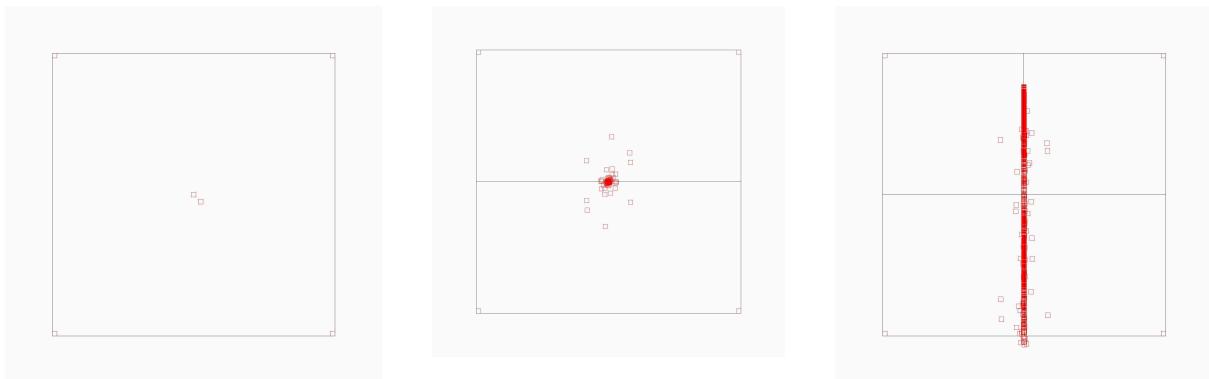


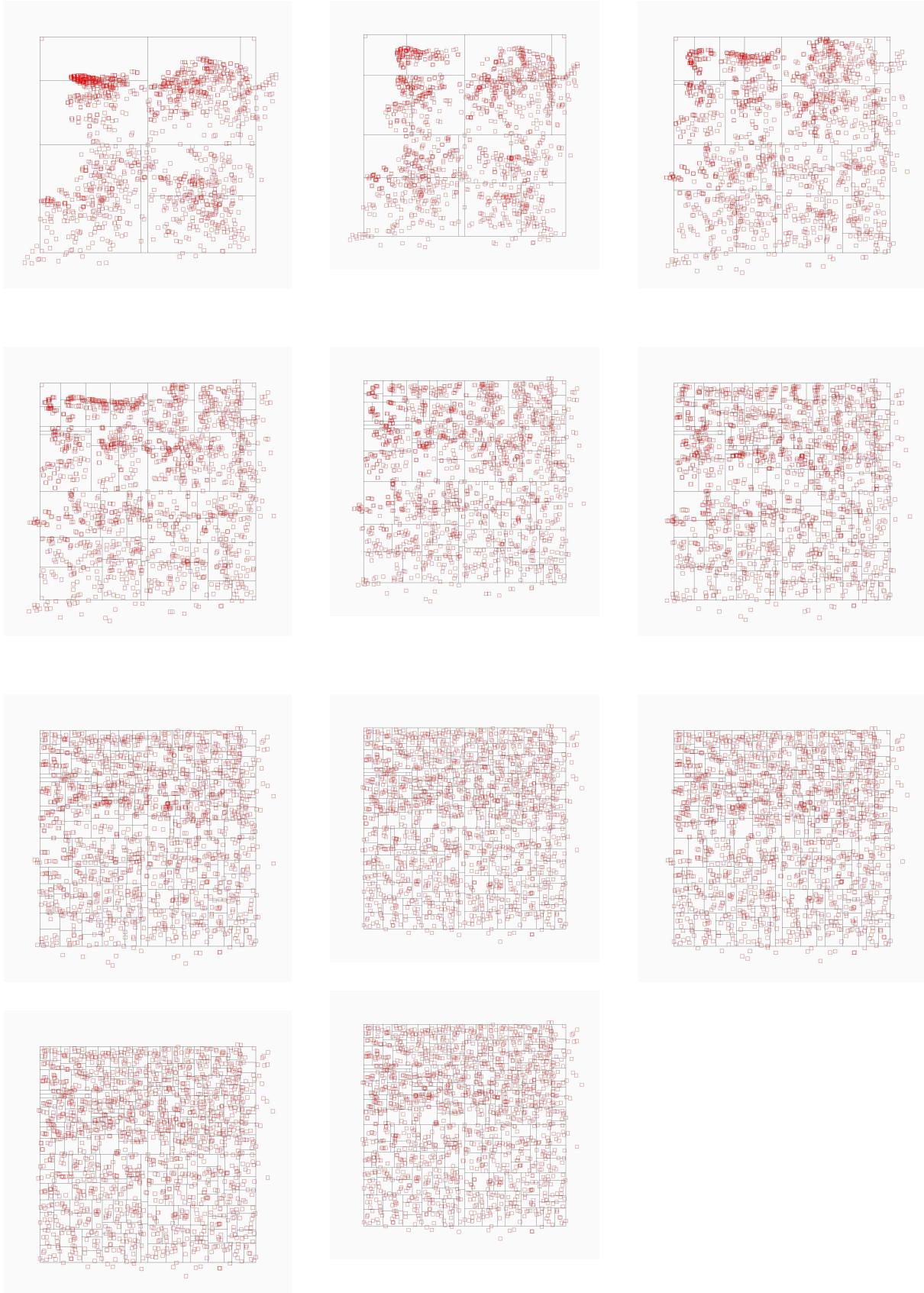


case2: netlist_H.txt

- without connection: partition plot 1~14

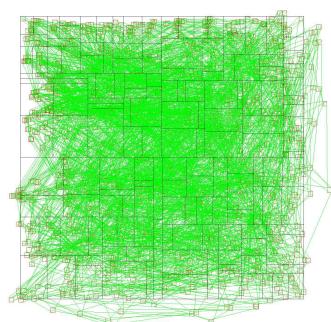
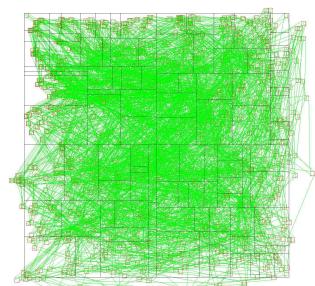
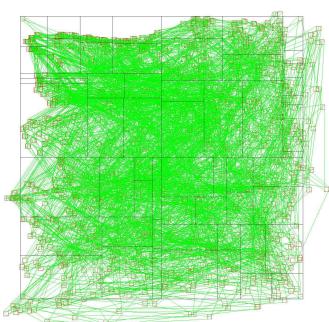
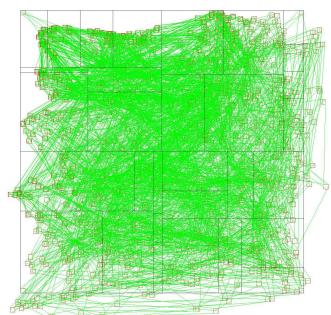
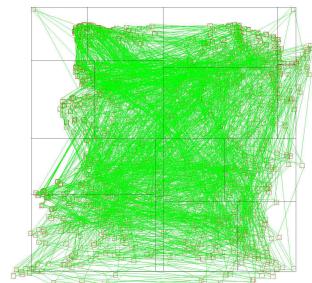
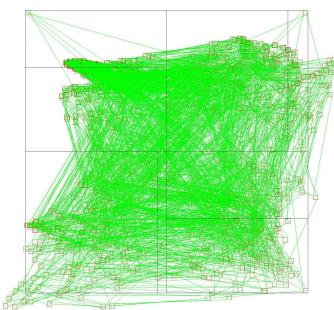
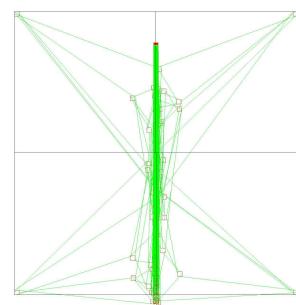
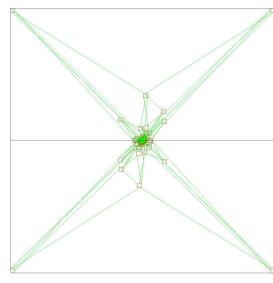
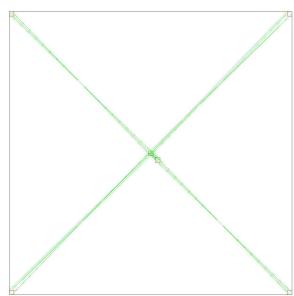
```
C:\Users\user\Documents\課程 X + | v
input the name of netlist file.(include file name extension)
netlist_H.txt
partition finish!
Overall iterations: 13
Overall wire length is 150339.625(include fixed cells)
Calculation end!
Time for Gordian iterations: 113.88
```

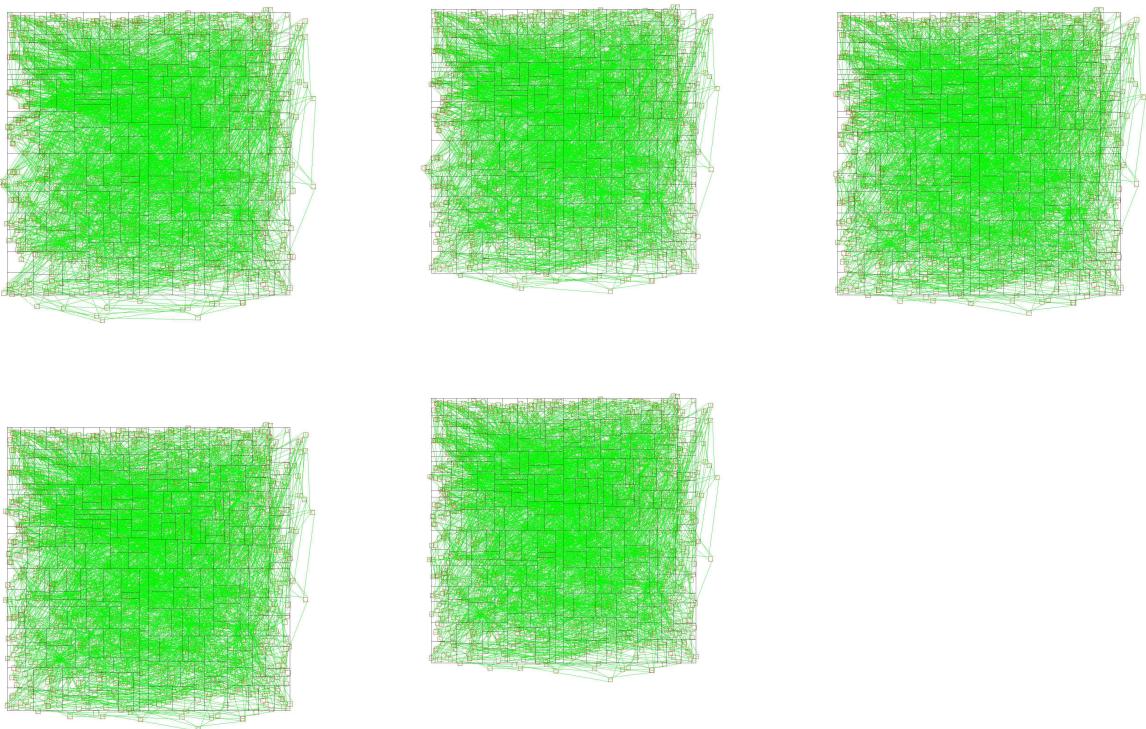




2. with connection: partition plot 1~14

```
C:\Users\user\Documents\課業 X + ▾  
input the name of netlist file.(include file name extension)  
netlist_H.txt  
partition finish!  
Overall iterations: 13  
Overall wire length is 150339.625(include fixed cells)  
Calculation end!  
Time for Gordian iterations: 280.20
```

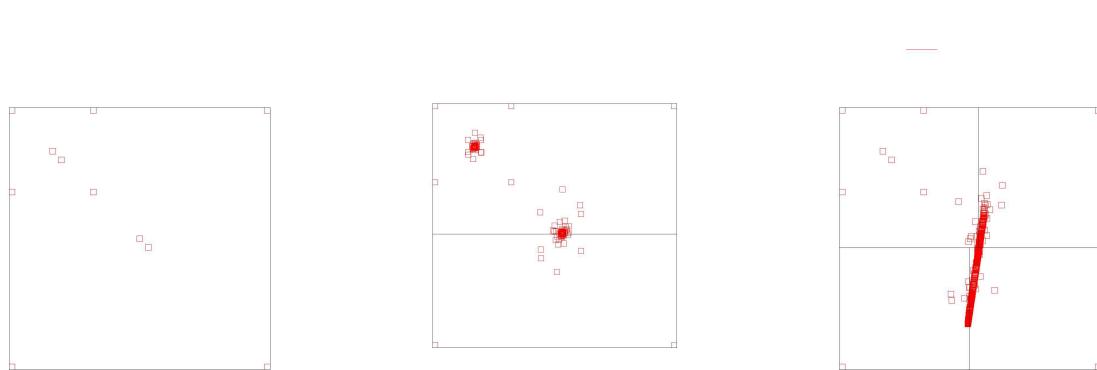


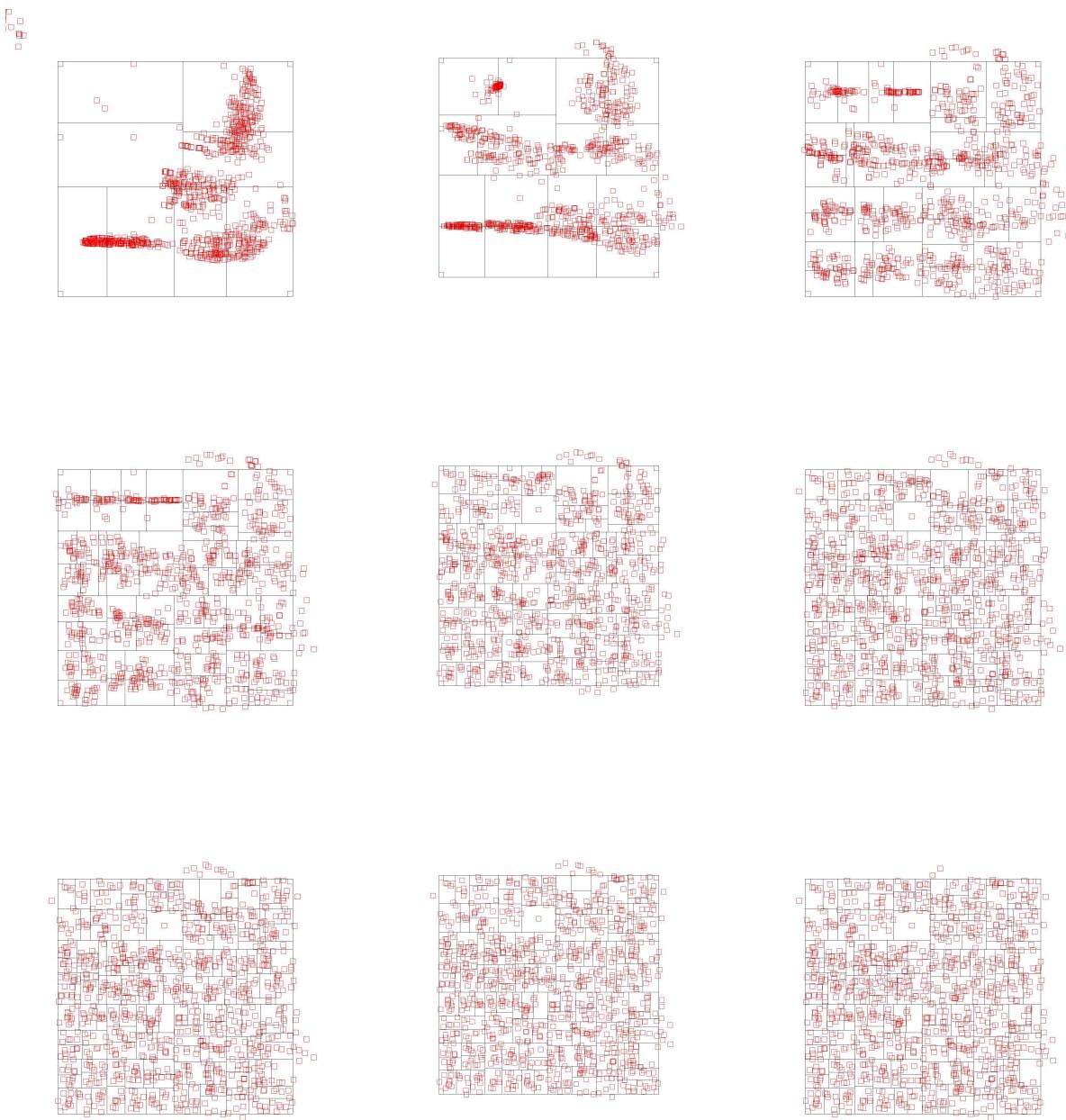


case3: netlist_A.txt (External netlist file)

- without connection: partition plot 1~12

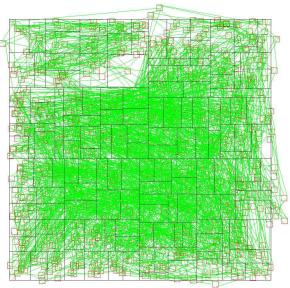
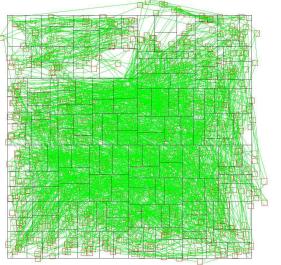
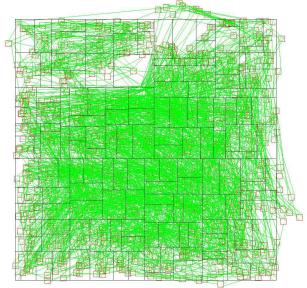
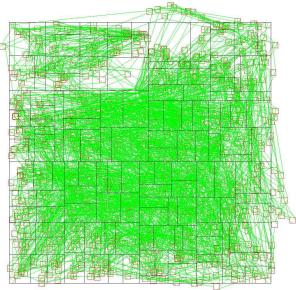
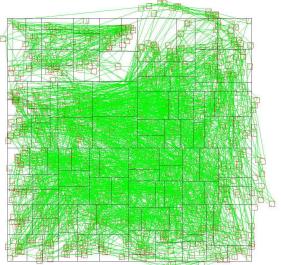
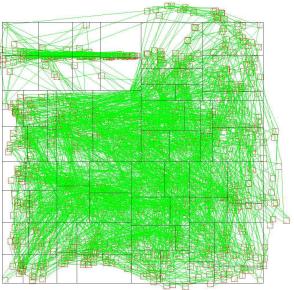
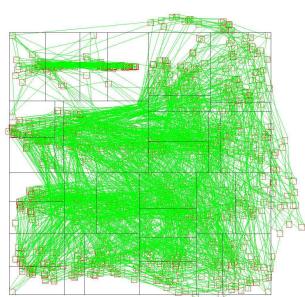
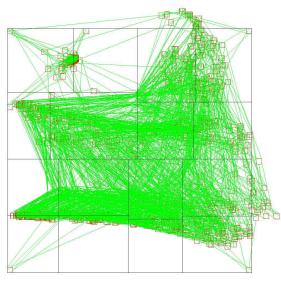
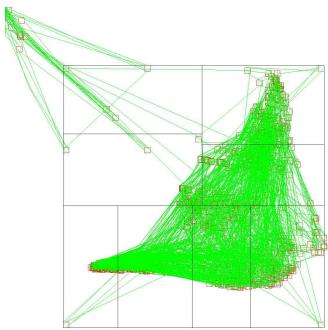
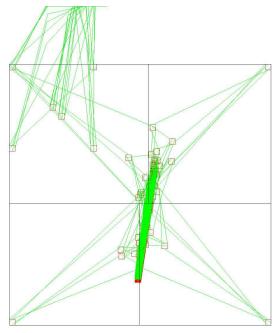
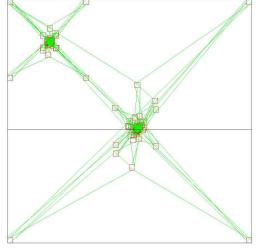
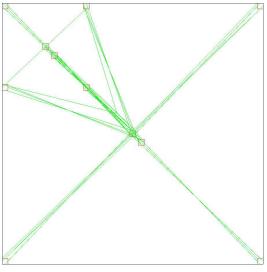
```
C:\Users\user\Documents\課程 x + v
input the name of netlist file.(include file name extension)
netlist_A.txt
partition finish!
Overall iterations: 11
Overall wire length is 72144.852(include fixed cells)
Calculation end!
Time for Gordian iterations: 21.10
```





2. with connection: partition plot 1~12

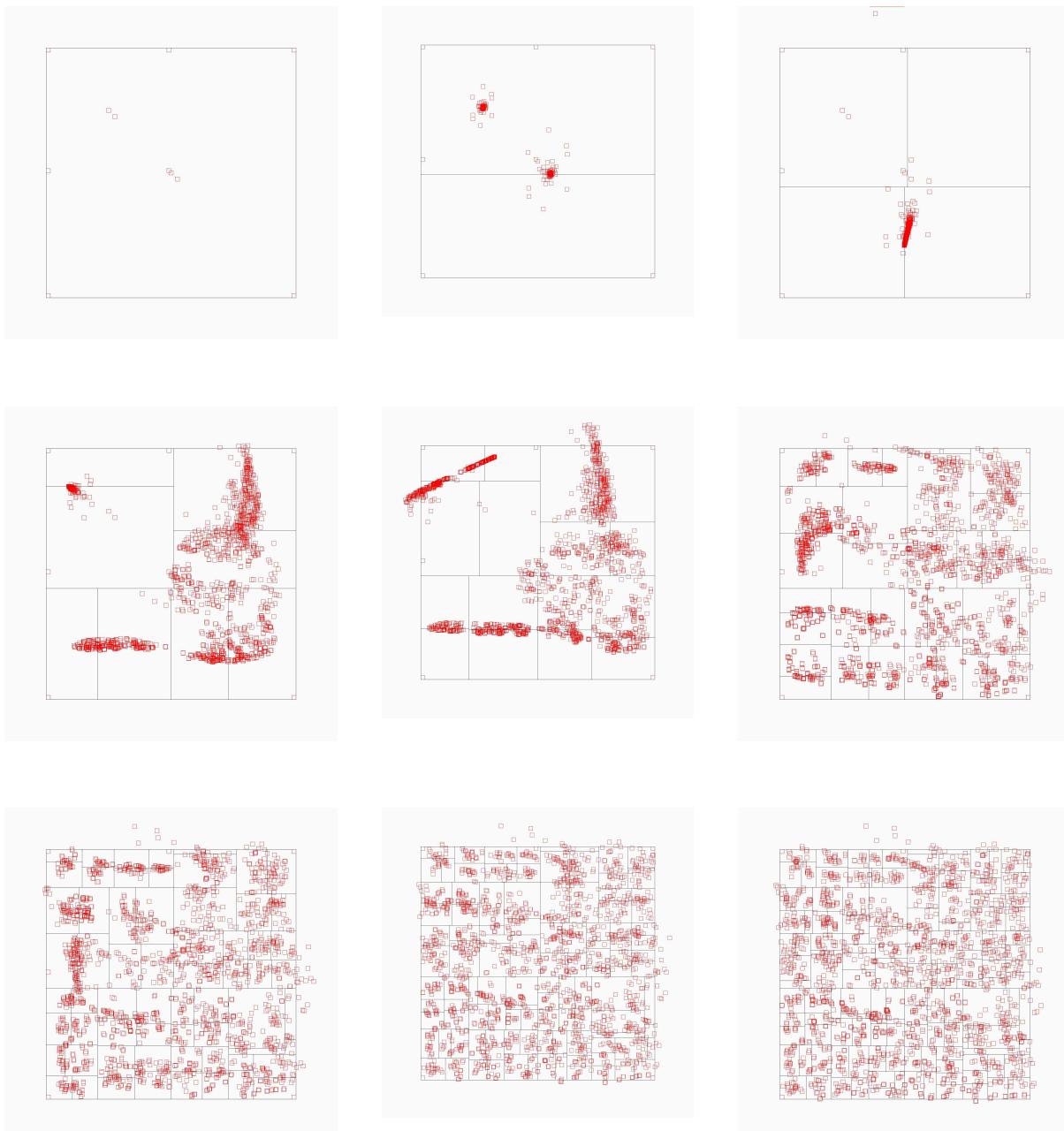
```
C:\Users\user\Documents\課程 X + ▾
input the name of netlist file.(include file name extension)
netlist_A.txt
partition finish!
Overall iterations: 11
Overall wire length is 72144.852(include fixed cells)
Calculation end!
Time for Gordian iterations: 56.70
```

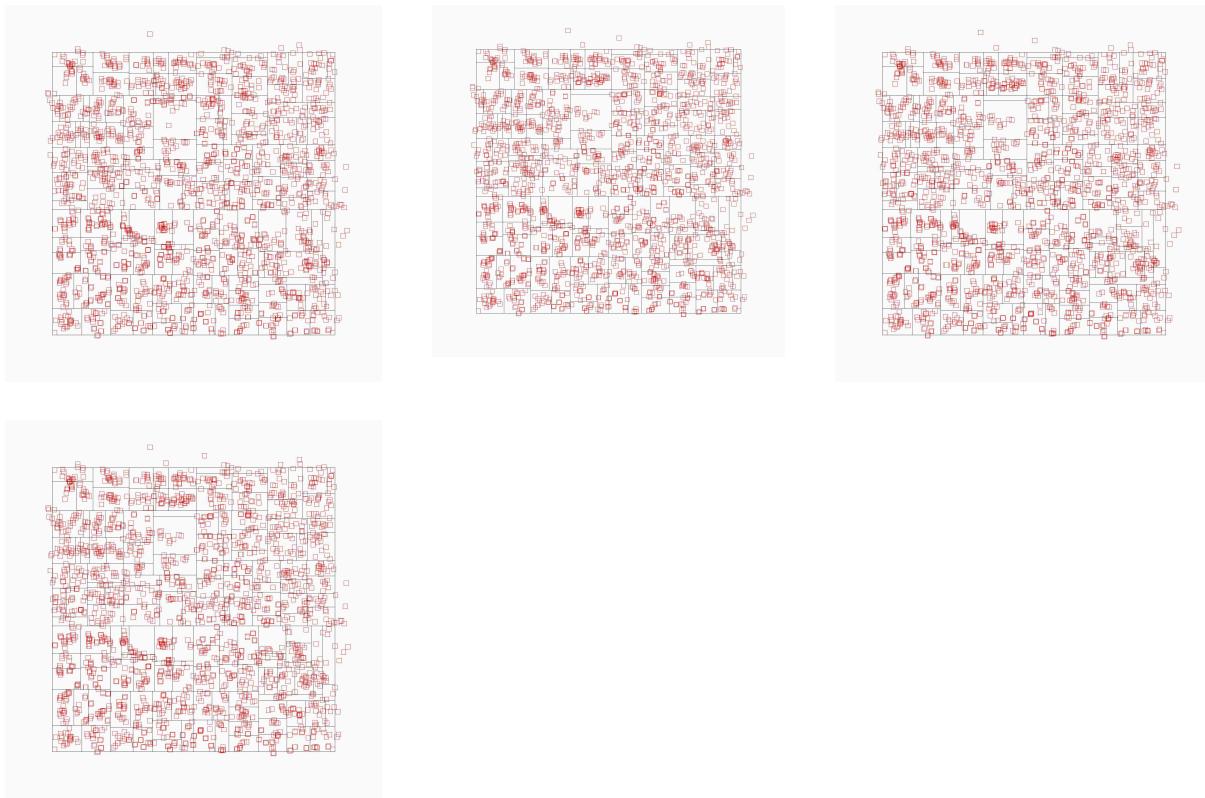


case4: netlist_B.txt (External netlist file)

- without connection: partition plot 1~13

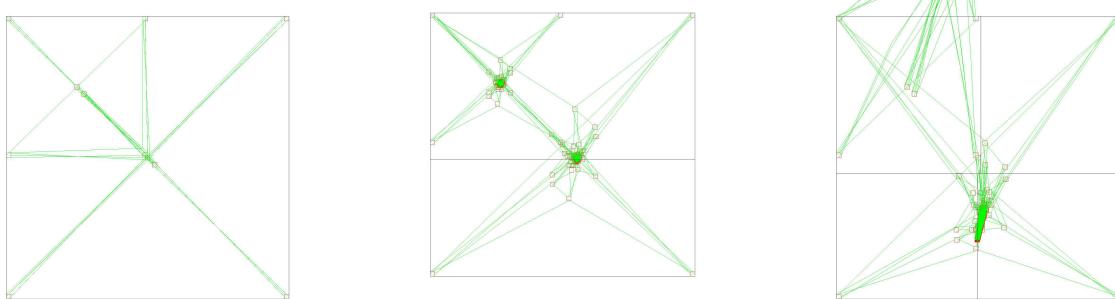
```
C:\Users\user\Documents\課業 X + ▾  
input the name of netlist file.(include file name extension)  
netlist_B.txt  
partition finish!  
Overall iterations: 12  
Overall wire length is 167382.875(include fixed cells)  
Calculation end!  
Time for Gordian iterations: 105.13
```

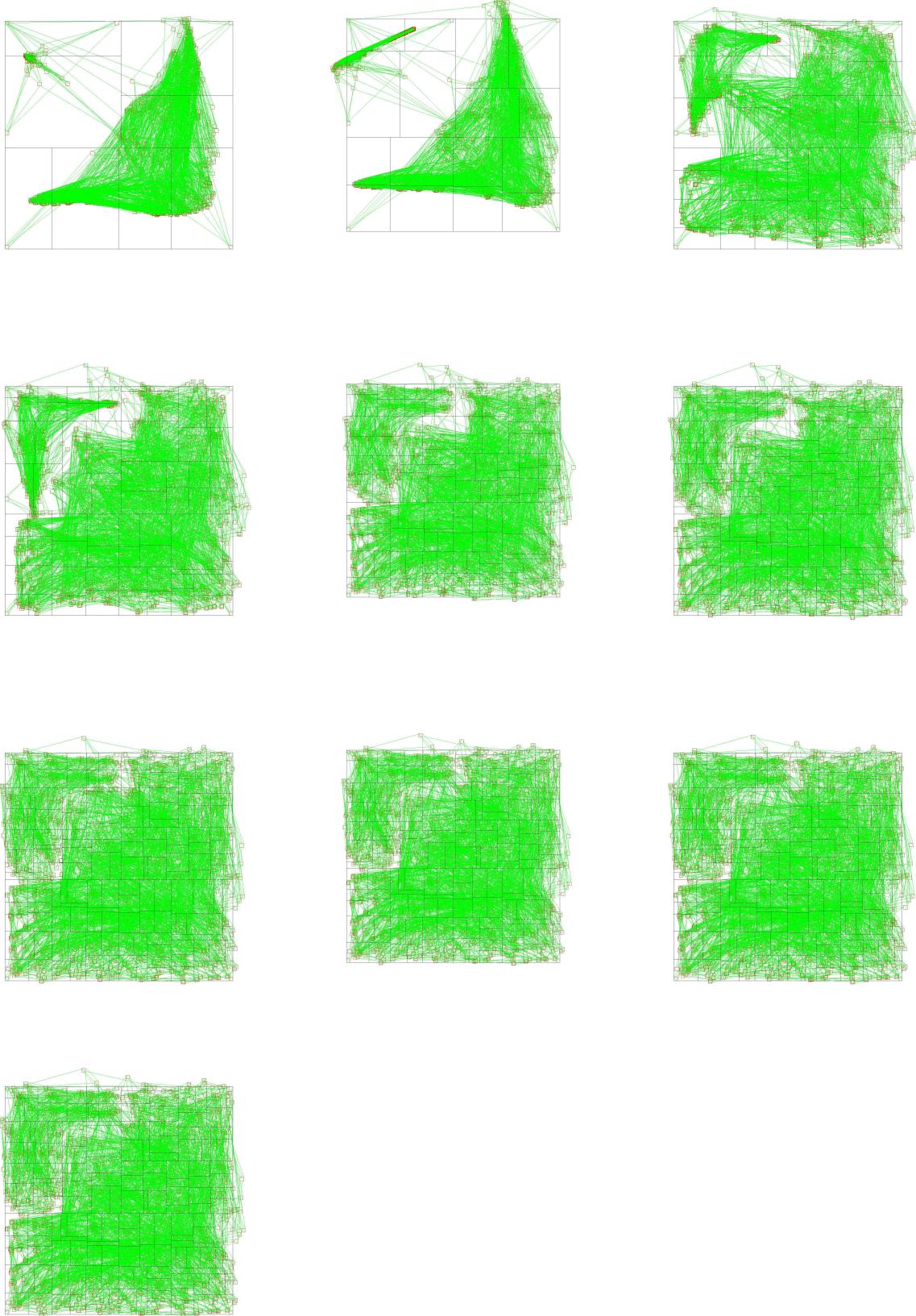




2. with connection: partition plot 1~13

```
C:\Users\user\Documents\課程 X + ▾
input the name of netlist file.(include file name extension)
netlist_B.txt
partition finish!
Overall iterations: 12
Overall wire length is 167382.875(include fixed cells)
Calculation end!
Time for Gordian iterations: 364.57
```





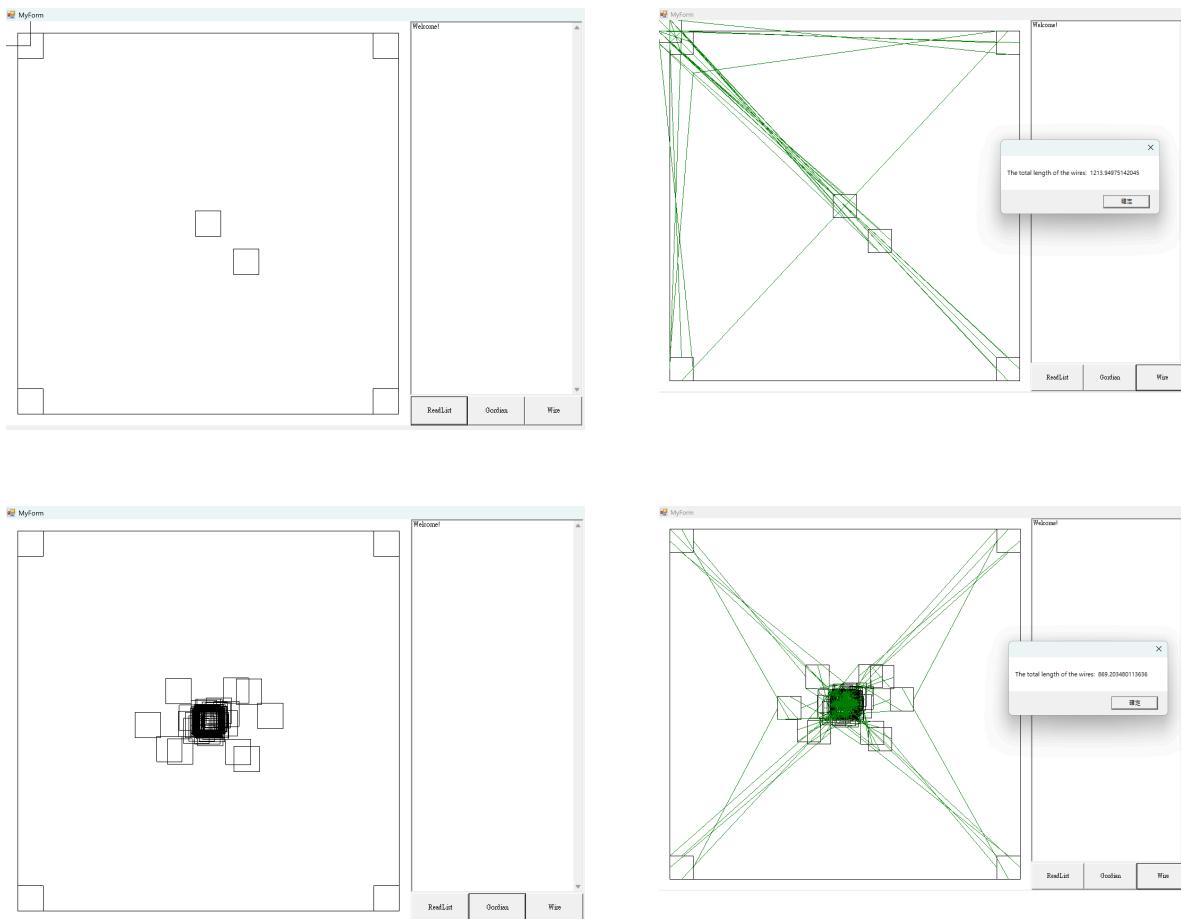
整理

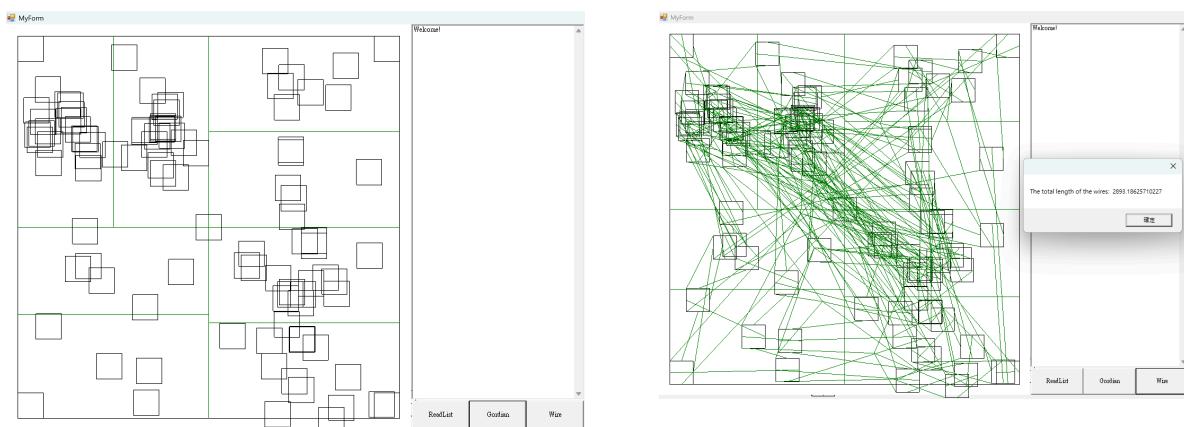
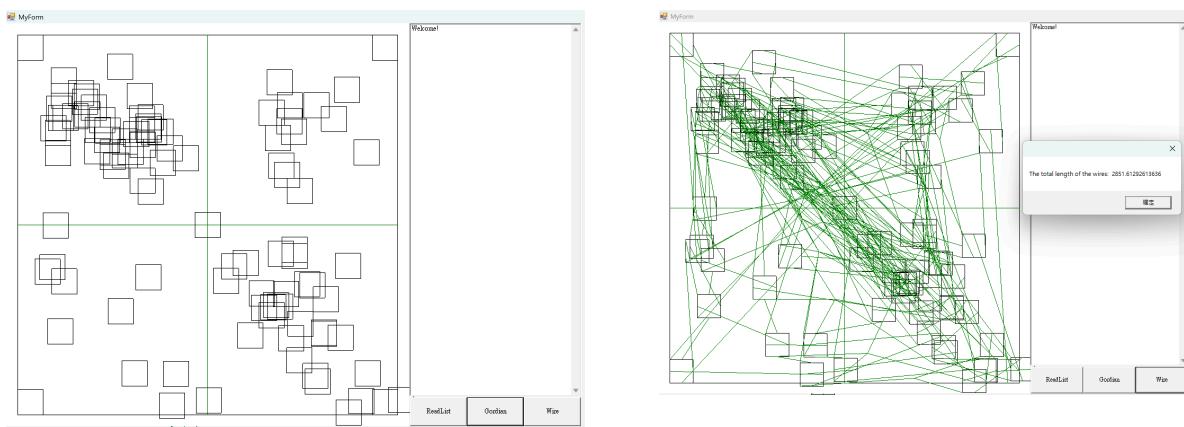
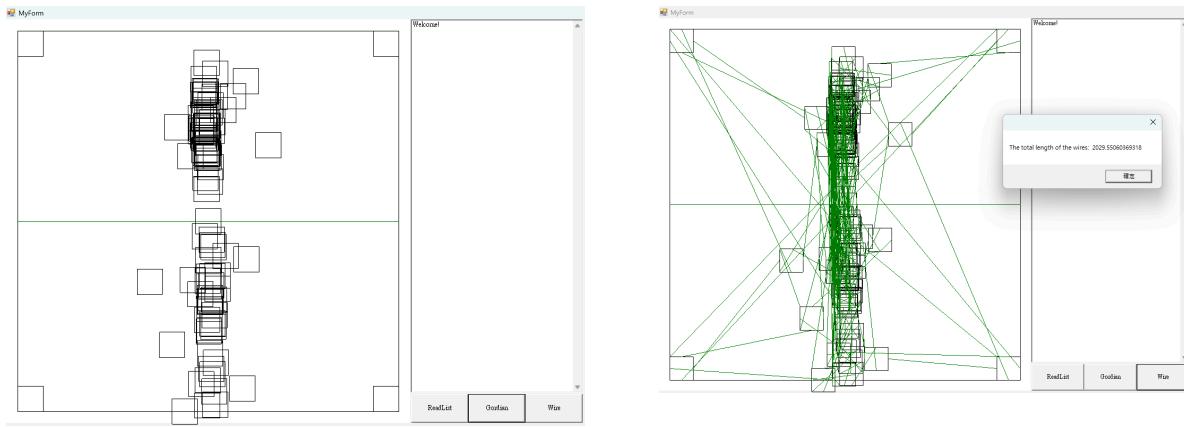
資料	Num Cells	lim_K	Num Partition	Total wire length	Iteration (s)
case1	400	5	10	20539.939	1.91
case2	1600	10	13	150339.625	113.88
case3	1000	10	11	72144.852	21.10
case4	2000	20	12	167382.875	105.13

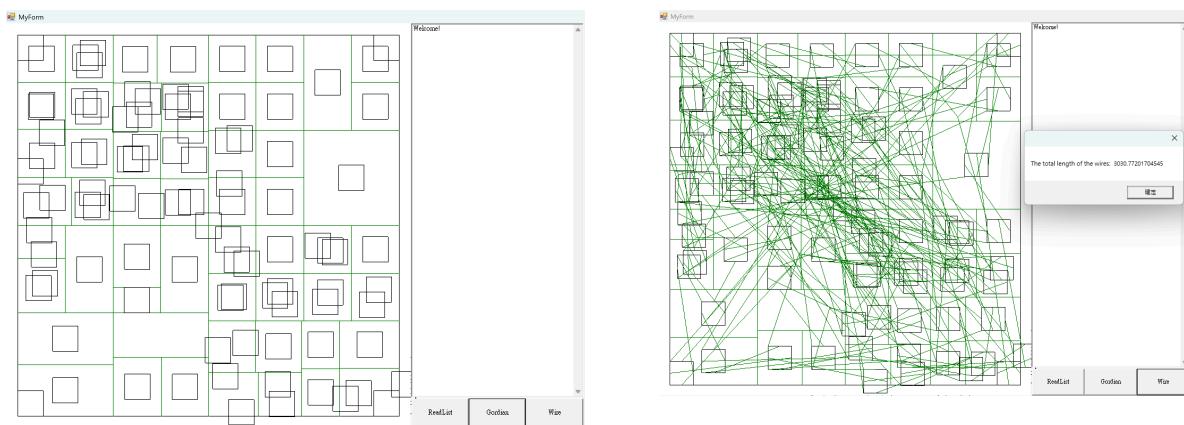
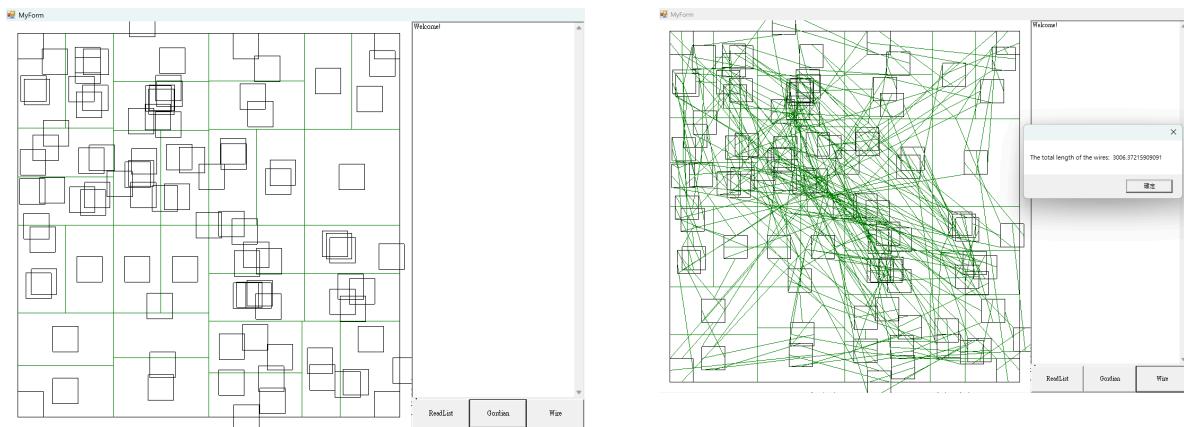
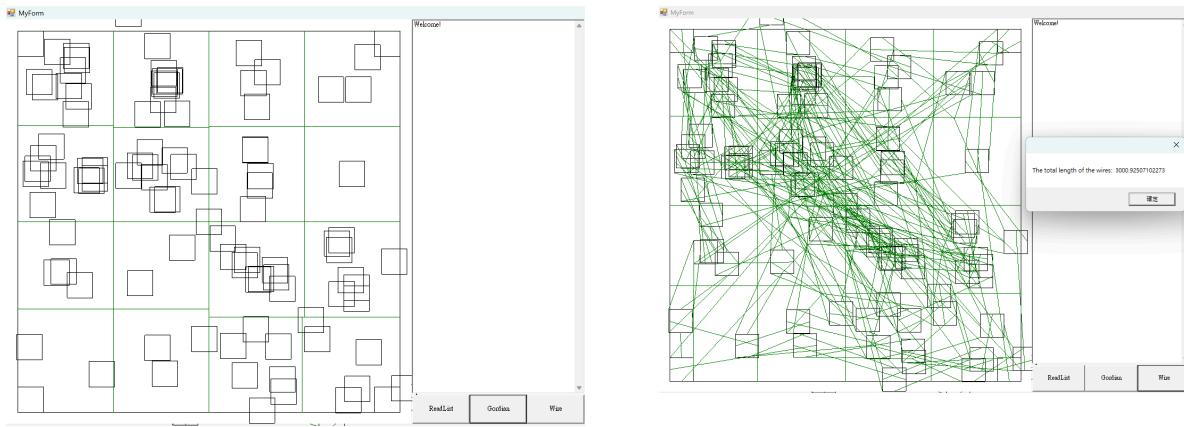
* lim_K 為最後partition完的結果，每一個region最多有幾個cell。

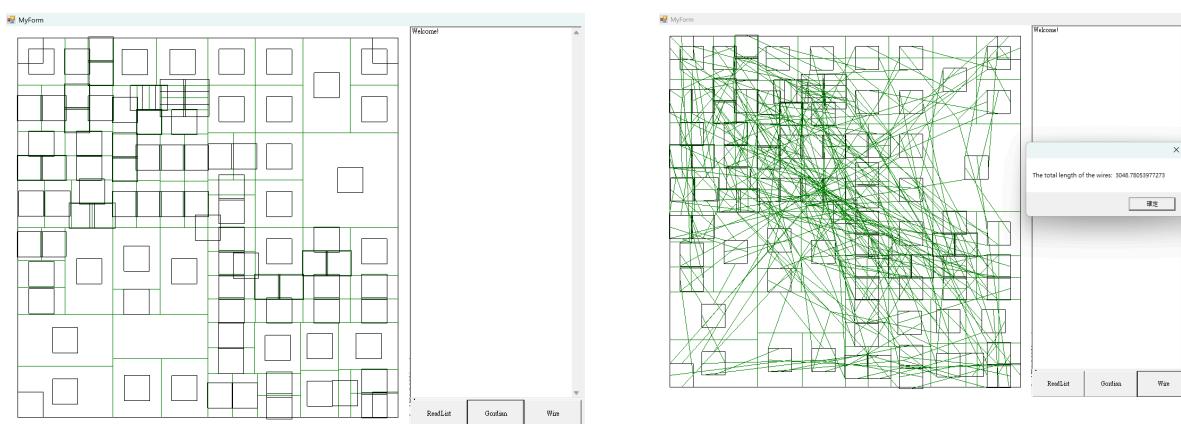
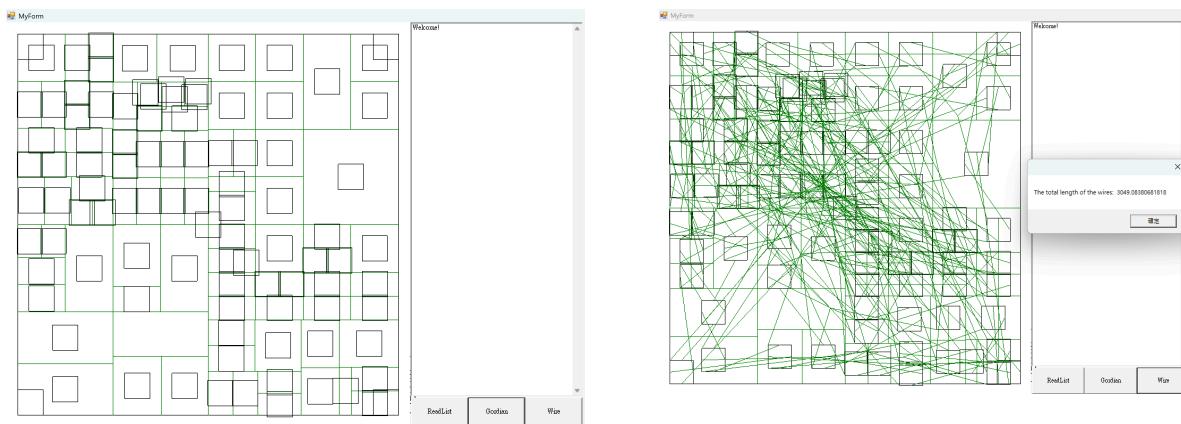
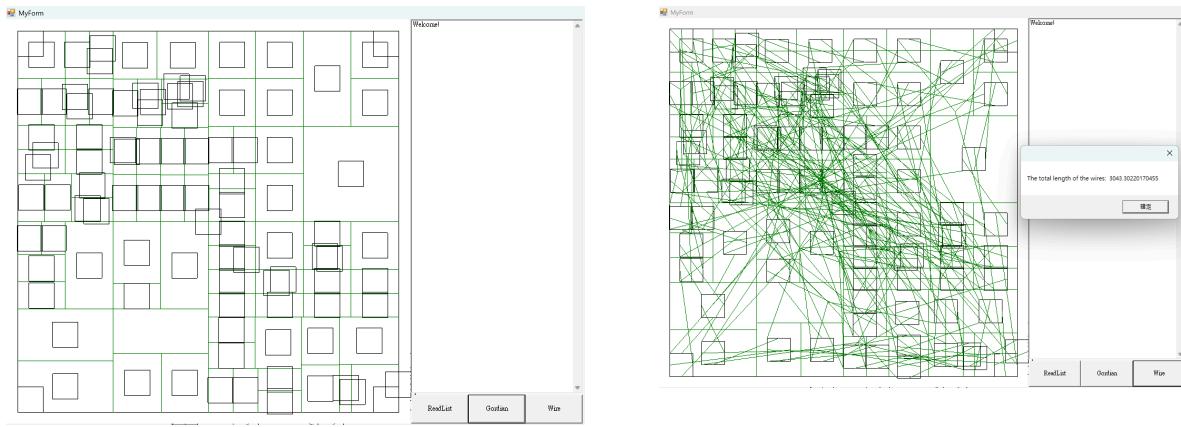
Result with metrics (Option B)

Case1: netlist_10%5b1%5d.txt





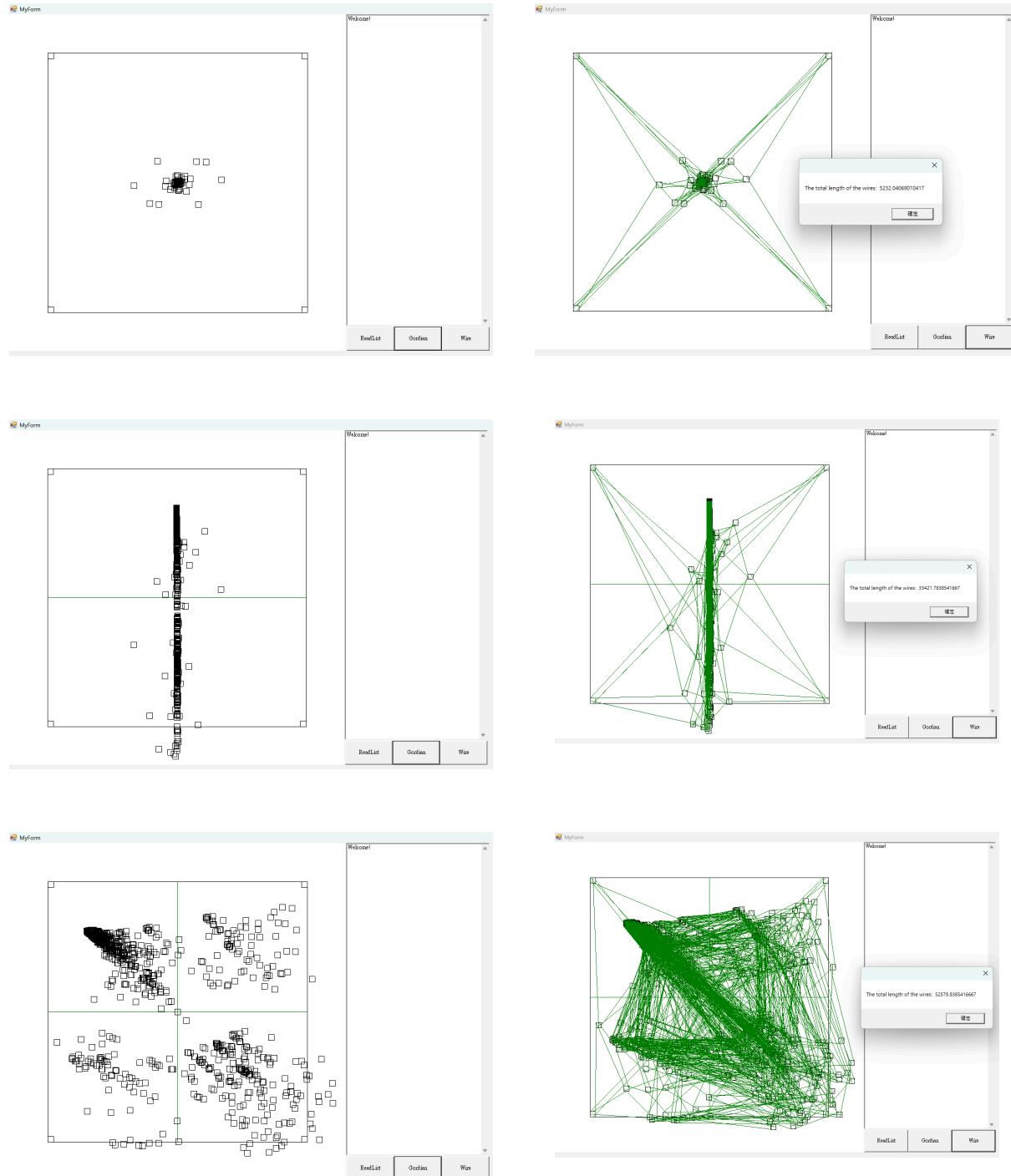


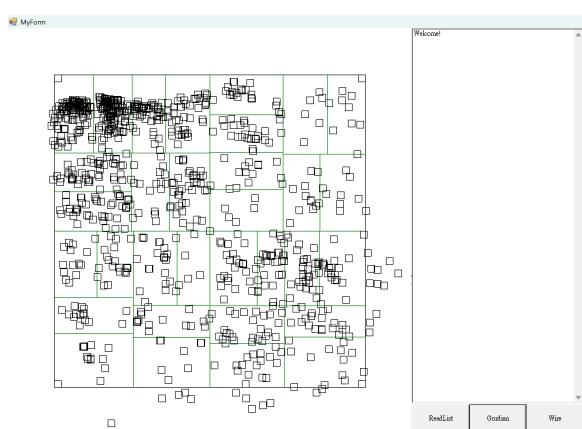
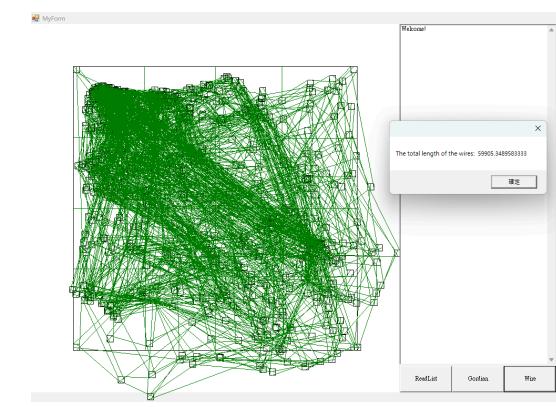
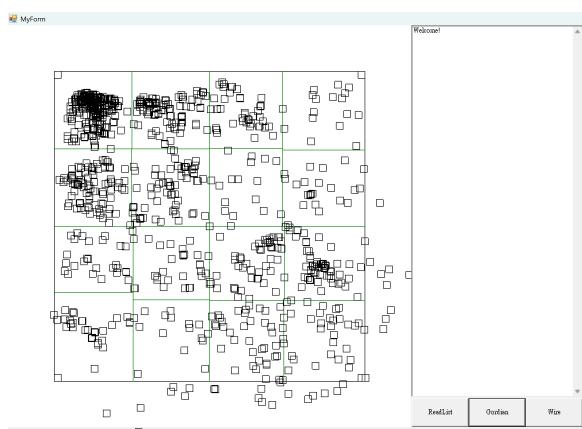
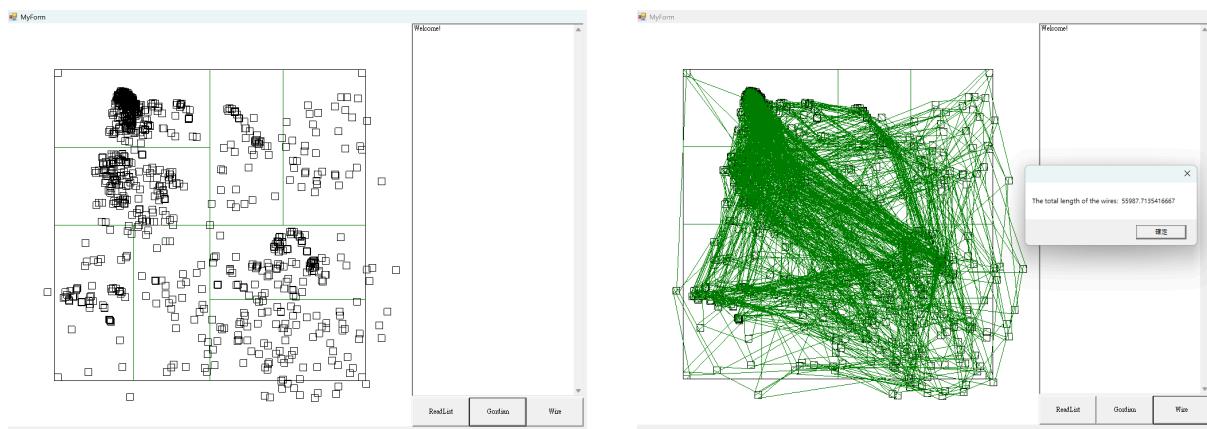


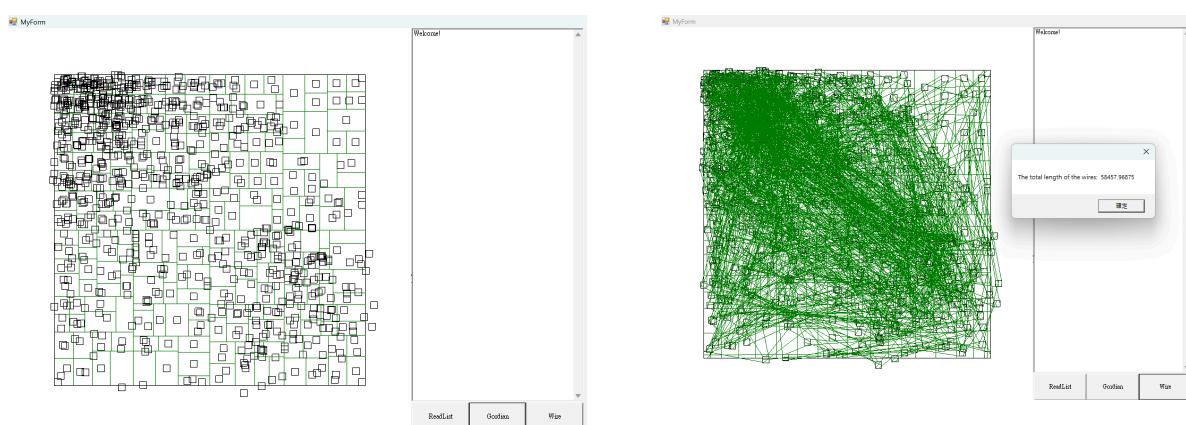
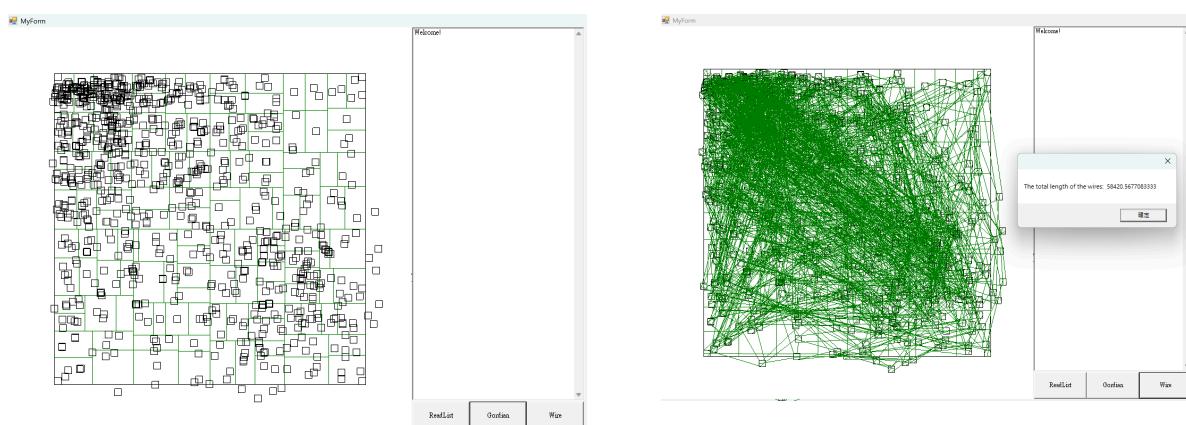
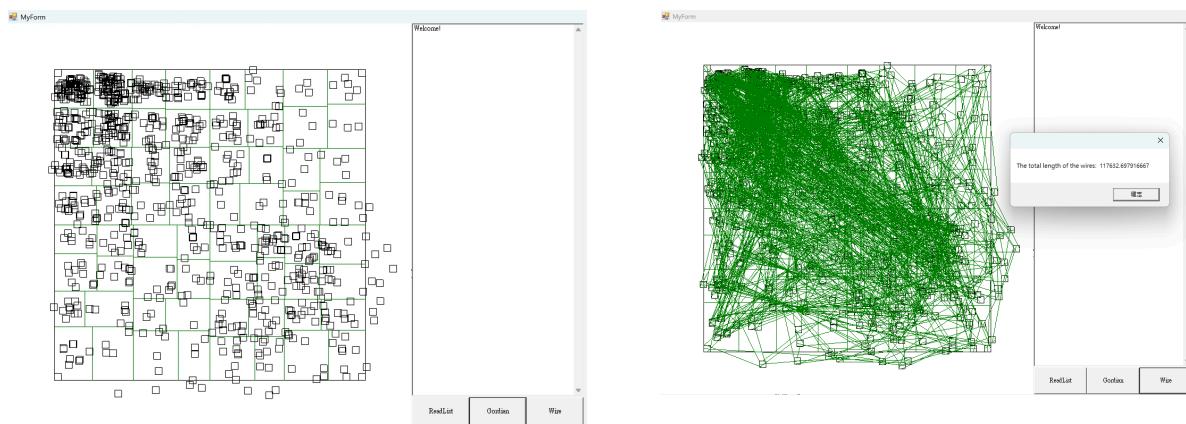
Case 1	Total wire length
Initial	1213.94975
Partition 1	869.20348
Partition 2	2029.55060
Partition 3	2851.61293
Partition 4	2893.18625
Partition 5	3000.92507

Partition 6	3006.37216
Partition 7	3030.77201
Partition 8	3043.30220
Partition 9	3049.08381
Partition 10	3048.78054

Case2: netlist_30%5b1%5d_class.txt



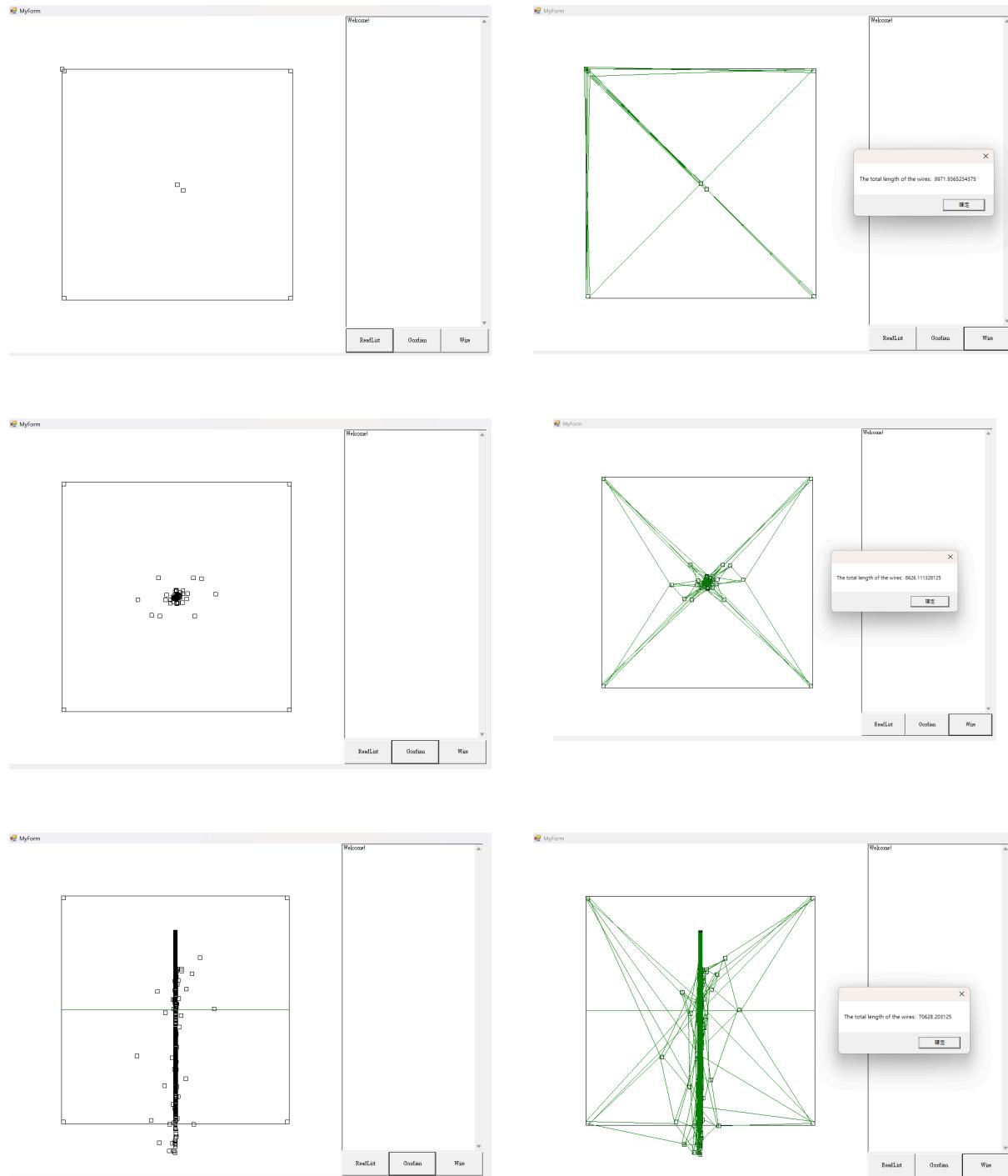


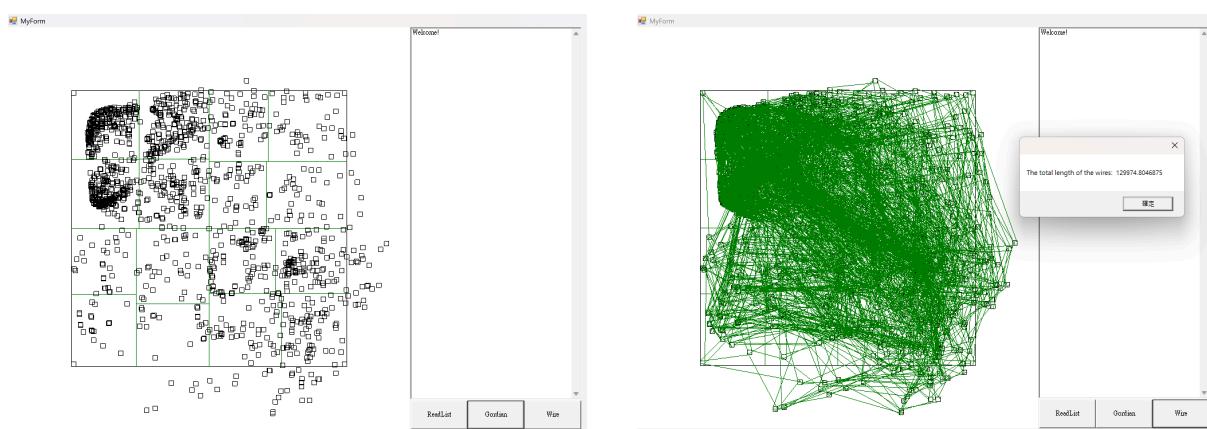
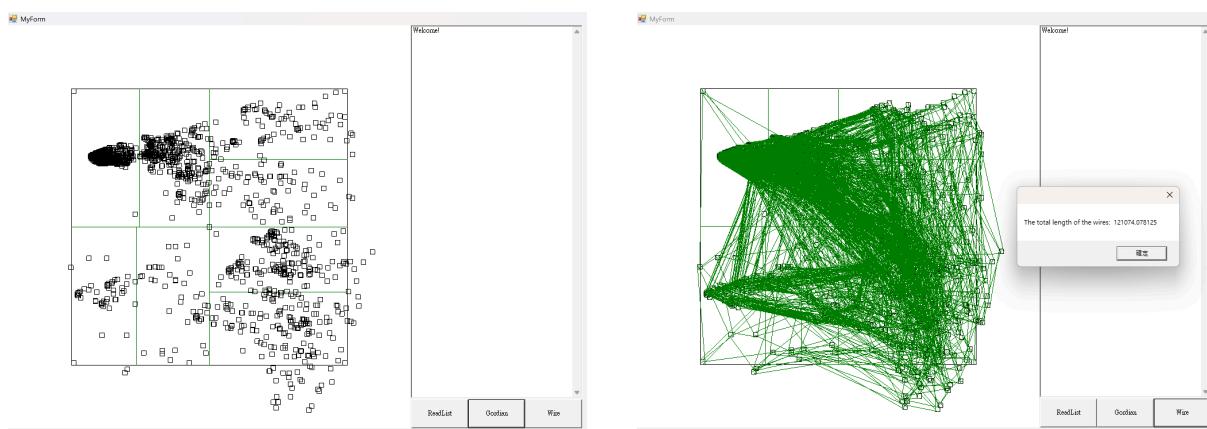
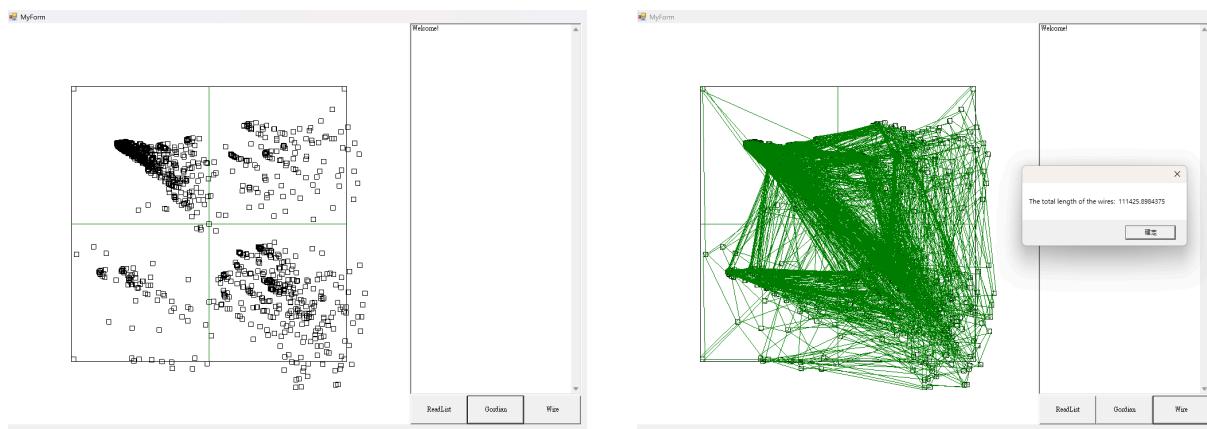


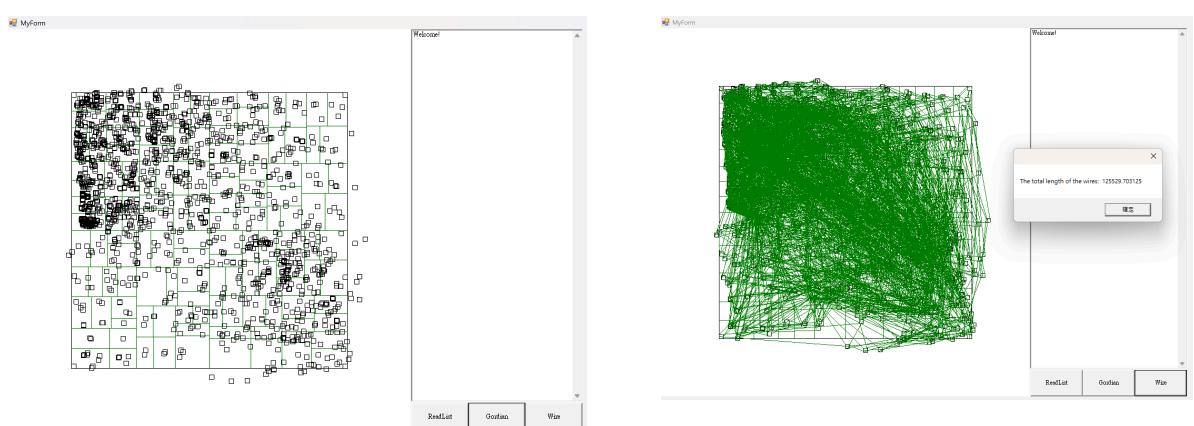
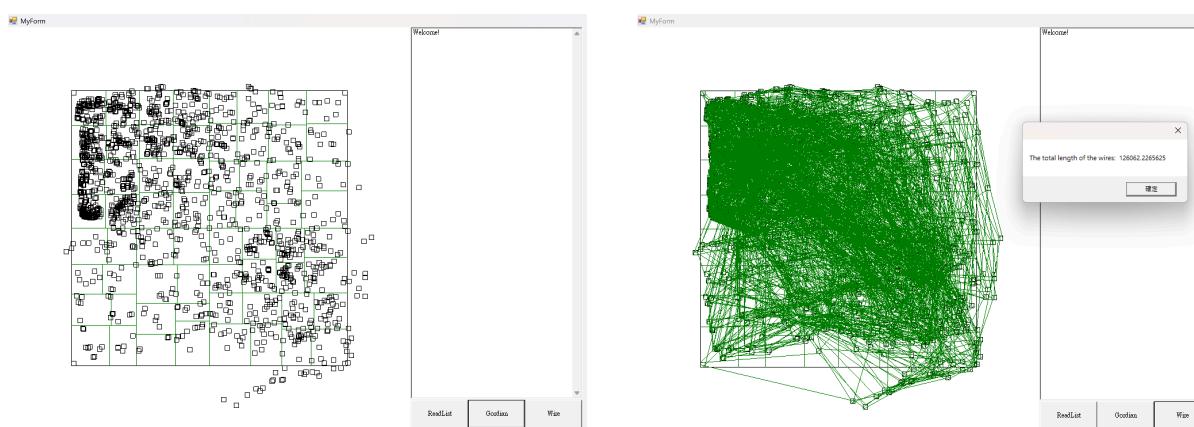
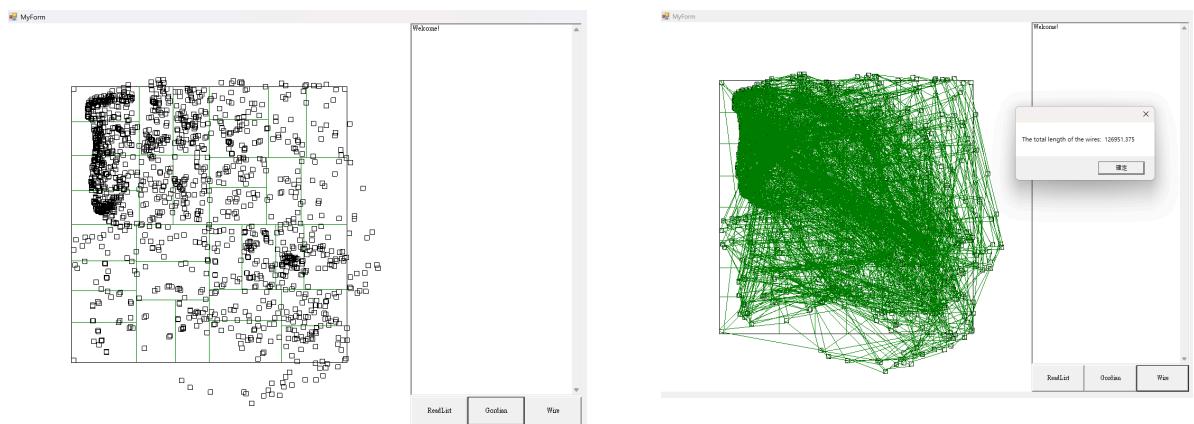
Case 2	Total wire length
Initial	5232.04069
Partition 1	33421.78385
Partition 2	52579.83854
Partition 3	55987.71354
Partition 4	59905.34895
Partition 5	58765.36458

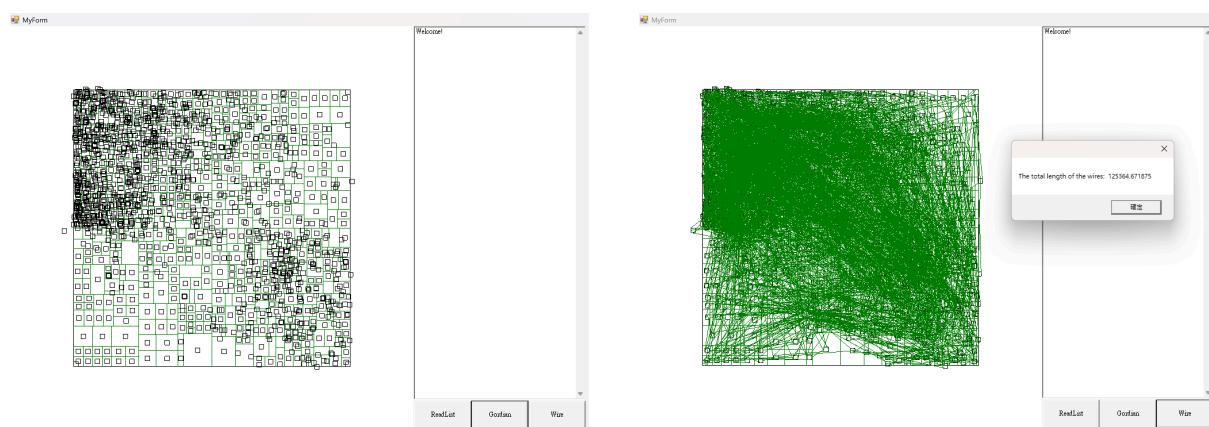
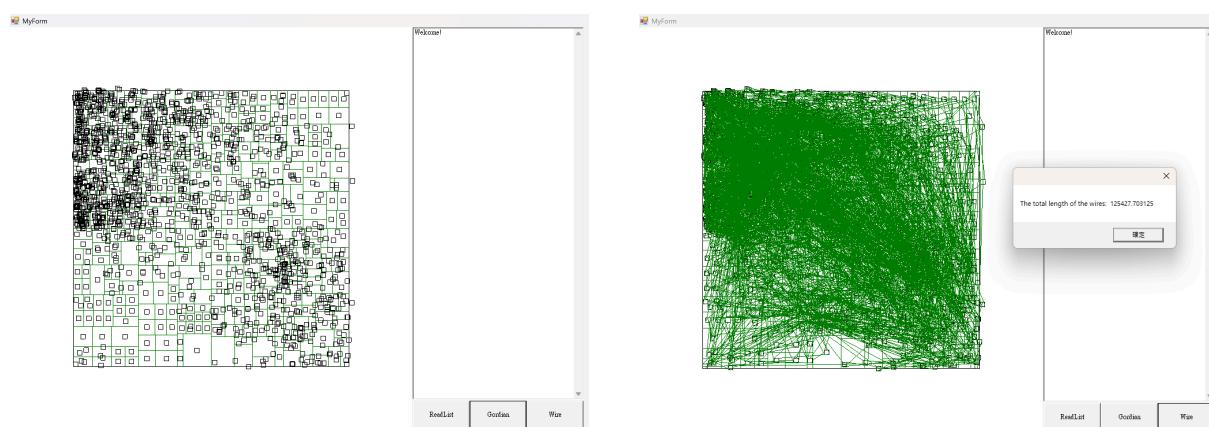
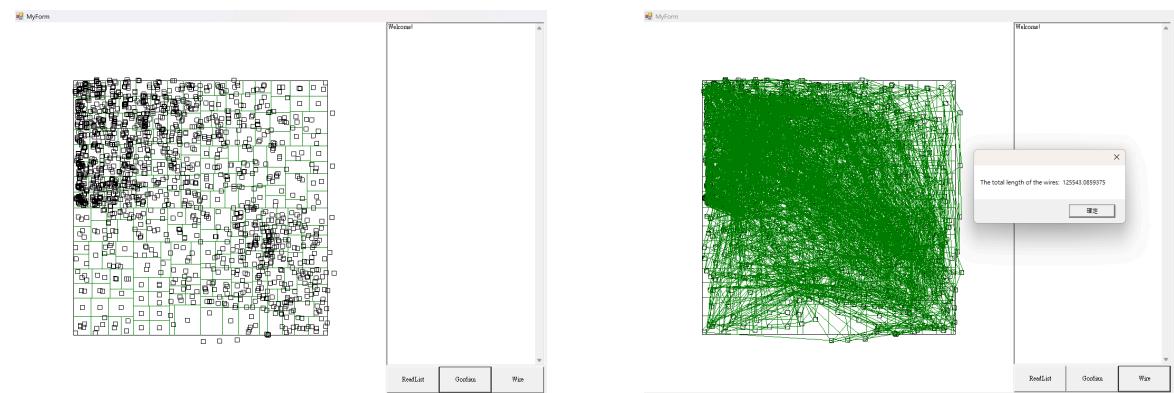
Partition 6	58816.34895
Partition 7	58420.56771
Partition 8	58457.96875

Case3: netlist_40%5b1%5d.txt





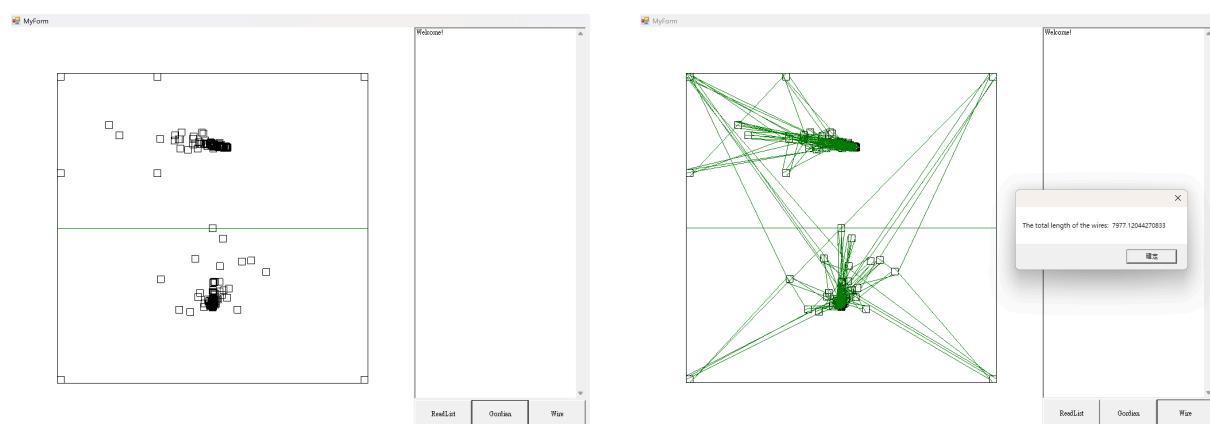
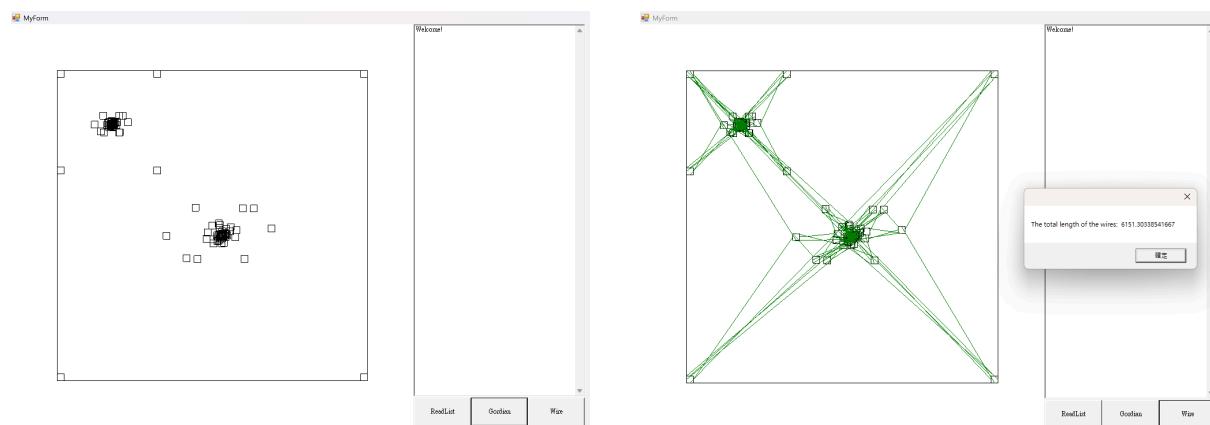
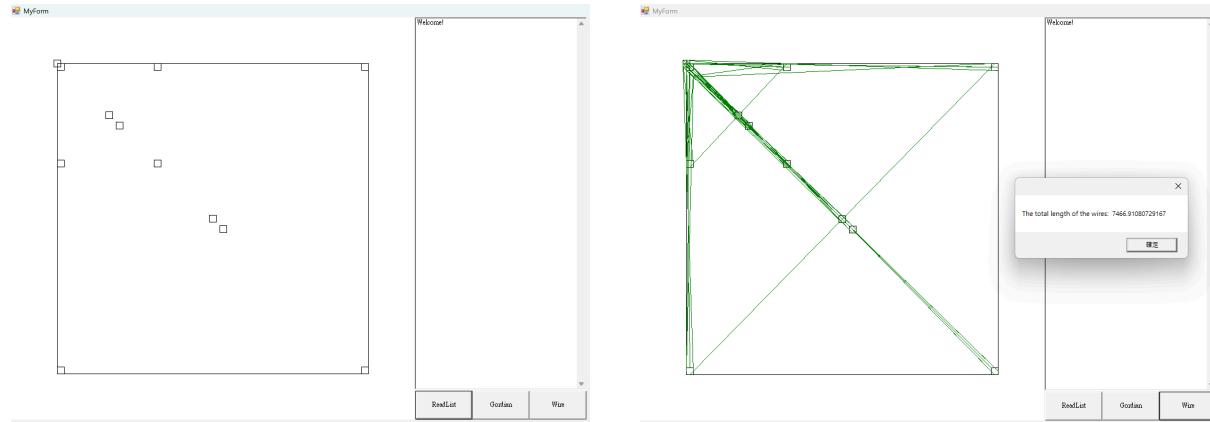


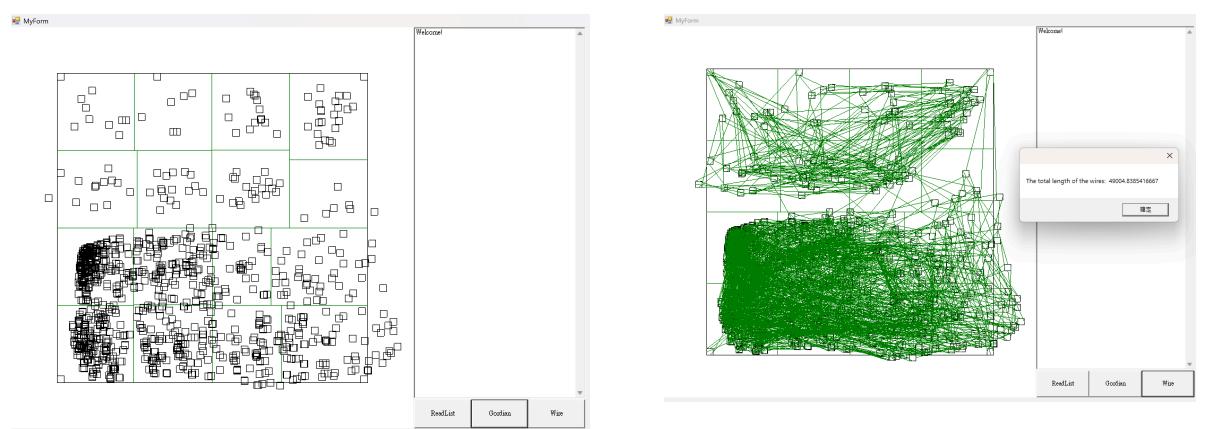
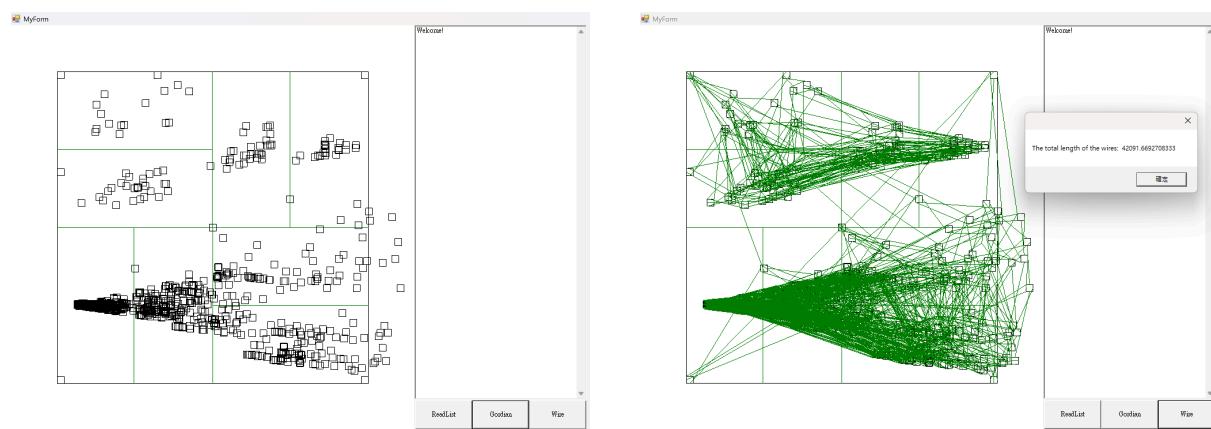
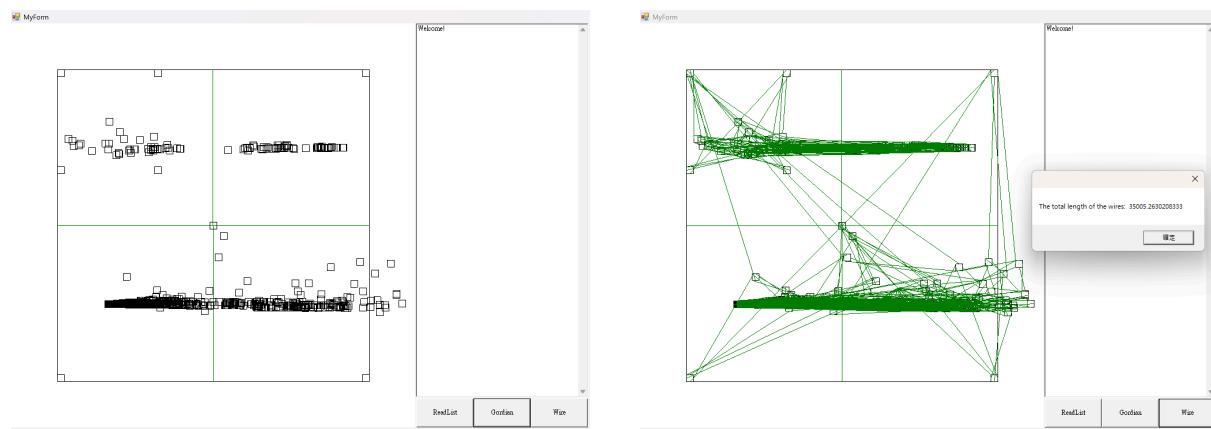


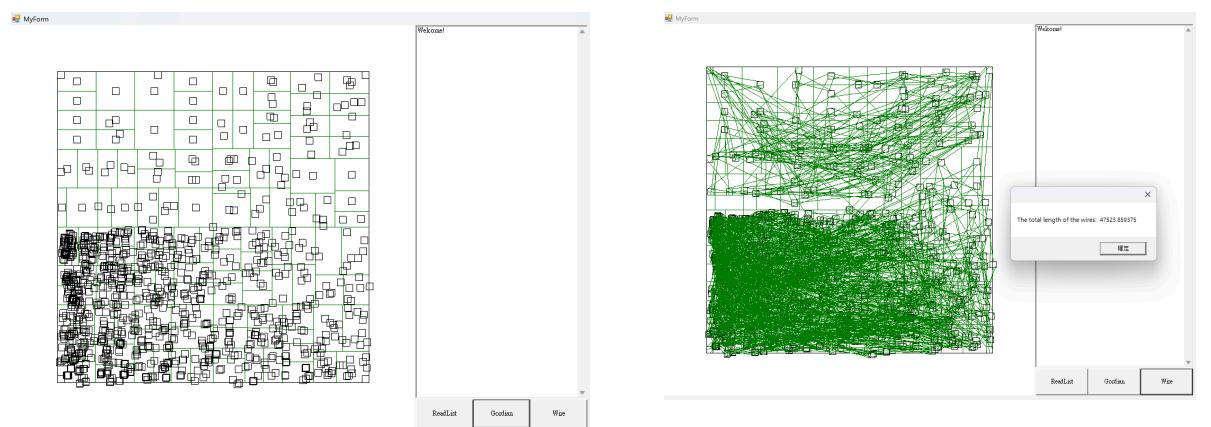
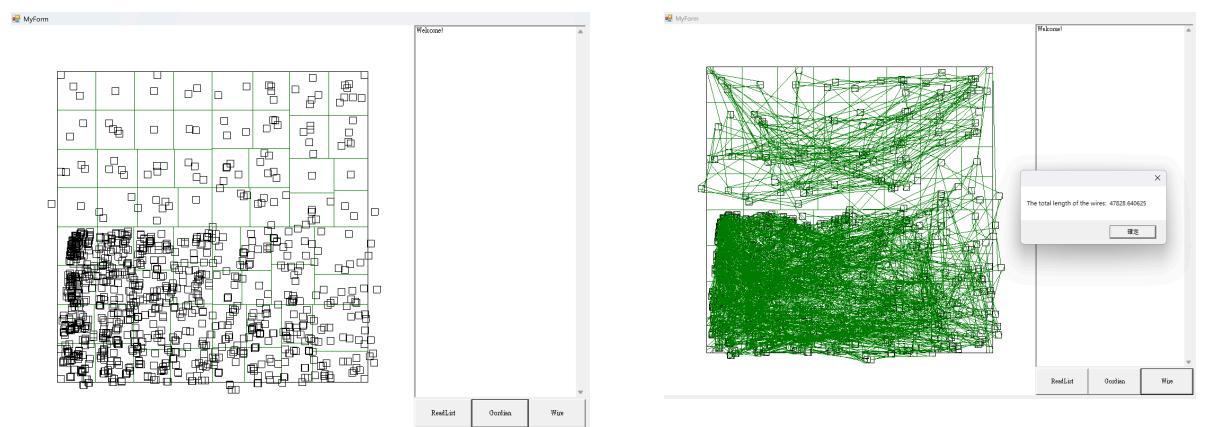
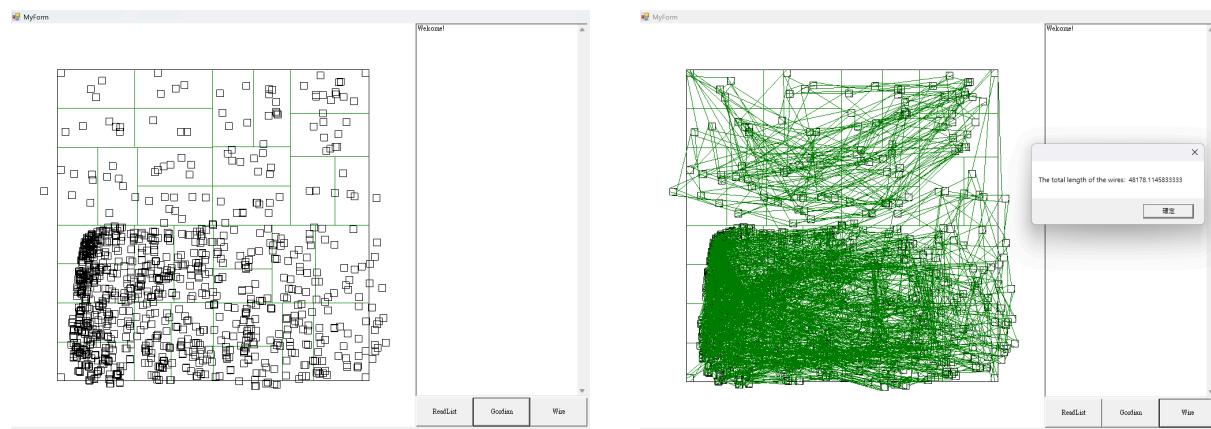
Case 3	Total wire length
Initial	9971.93652
Partition 1	8626.11132
Partition 2	70628.20312
Partition 3	111425.89843
Partition 4	121074.07812
Partition 5	129974.80469
Partition 6	126951.375

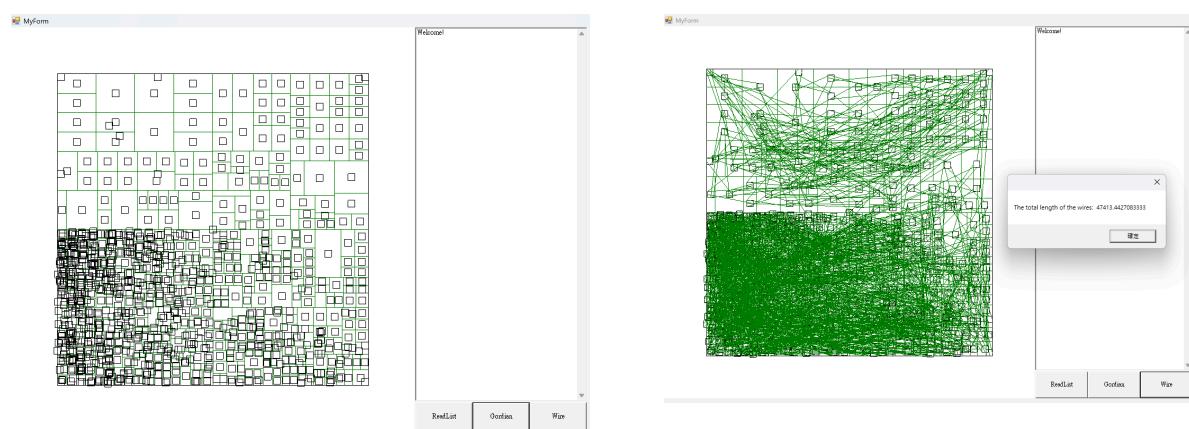
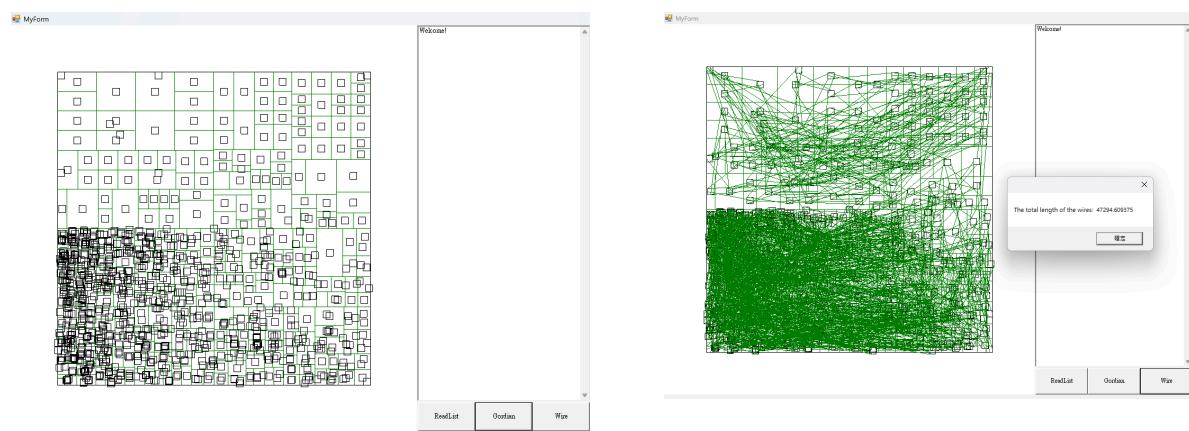
Partition 7	126062.22656
Partition 8	125529.70312
Partition 9	125543.08593
Partition 10	125427.70312
Partition 11	125364.67818

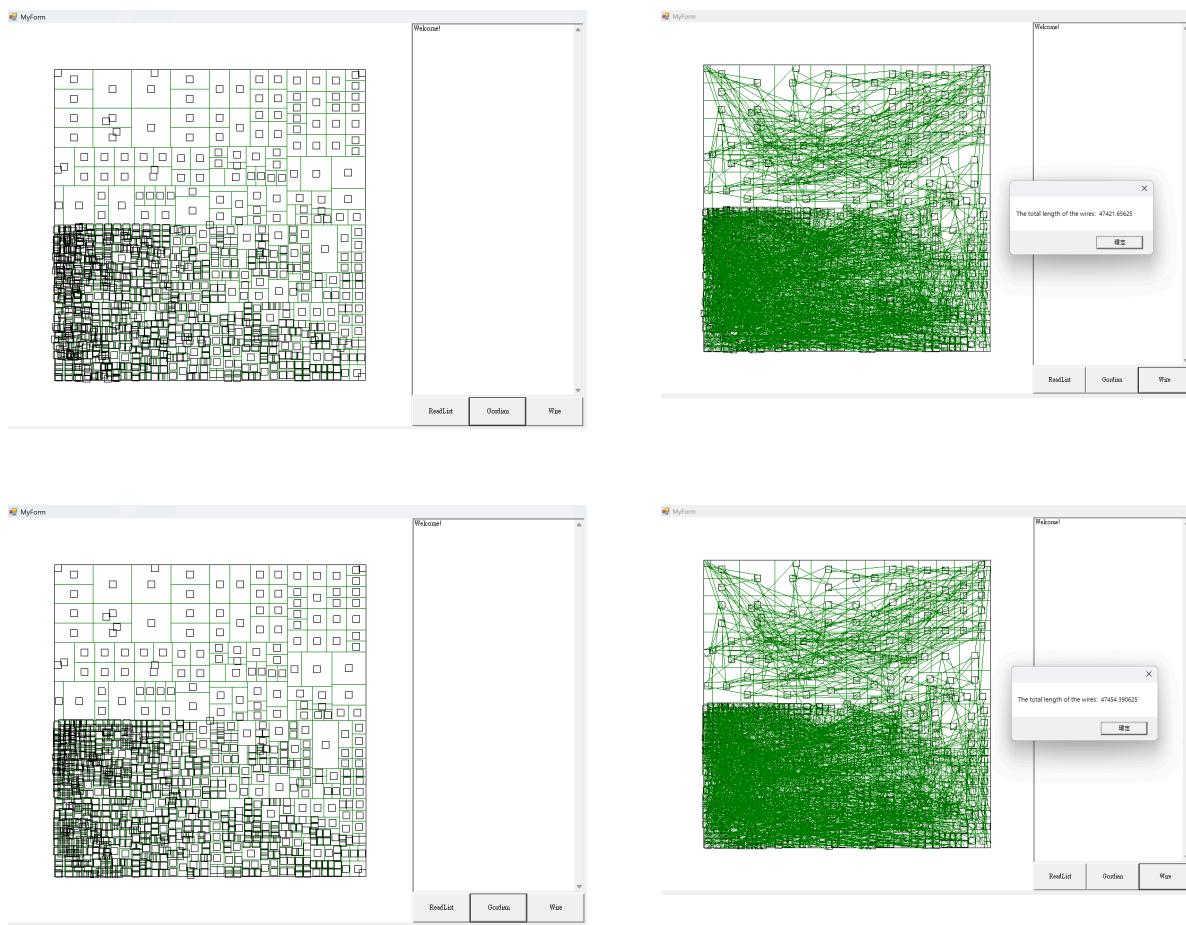
Case4: netlist_A.txt (External netlist file)











Case 4	Total wire length
Initial	7466.91080
Partition 1	6151.30338
Partition 2	7977.12044
Partition 3	35005.26302
Partition 4	42091.66927
Partition 5	49004.83854
Partition 6	48178.11458
Partition 7	47828.64062
Partition 8	47523.85937
Partition 9	47391.94791
Partition 10	47294.60937
Partition 11	47413.44271
Partition 12	47421.65625
Partition 13	47454.39062

整理

資料	Num Cells	Num Partition	Final wire length

case1	100	10	3048.78054
case2	900	8	58457.96875
case3	1600	11	125364.67818
case4	1000	13	47454.39062

Modules analysis

void Gordian::processFunc()

整個 program 的主要流程

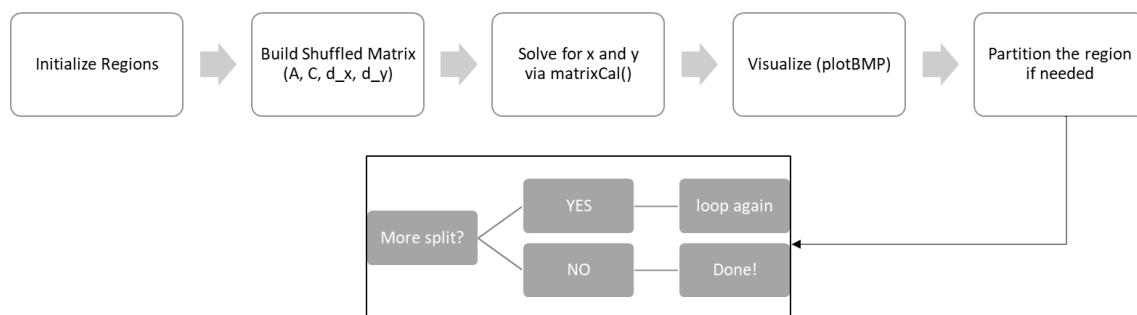
```
void Gordian::processFunc()
{
    netlist_read();      // 讀入 netlist 檔案，建構 Cells 與 Nets 的資料結構
    reCalCellID();       // 重新標記 moveable cell 的 ID
    set_up_obj();        // 建立線性代數的係數矩陣與向量 (C matrix, d_x, d_y)
    timer[0] = clock();
    beforShuffle();     // 初始化初始區域、中心點與分割參數
    shuffle_();          // 核心演算法，迭代式進行 cell 的位置優化與區域分割
    timer[0] = clock() - timer[0];

    printf("Time for Gordian iterations: %.2f\n", ((float)timer[0]) / CLOCKS_PER_SEC);
}
```

void Gordian::shuffle_()

◆ 主要流程圖：

```
while (1)
{
    1. 為每個區域統計 Cell 的資訊
    2. 建立打亂 (shuffled) 的矩陣 A、C、d_x、d_y
    3. 轉換成聯立方程式並解出 cell 的 x, y 座標
    4. 根據結果畫圖 (plotBMP)
    5. 根據分割條件進行區域切割 (partition)
    6. 若已無法再切割，跳出 while 迴圈
}
```



◆重要名詞：

1. A 矩陣：權重矩陣 (Weight Matrix)

- 根據網路 (net) 連線產生的，每個非固定 cell 是一個變數 (x、y 方向)
- A 是對所有 cell 做的「拉力計算」，衡量每對 cell 之間有多少「牽引力」。



舉例：3 個 movable cells A, B, C，其中 A 與 B 相連，B 與 C 相連（兩個 net）
那

A 會是類似這樣的矩陣（簡化為對稱形式）：

$$A = \begin{bmatrix} 1 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{bmatrix}$$

類似 Laplacian matrix，反映了「每個 cell 對自己與其他 cell 的拉扯力」。

2. C 矩陣：區塊矩陣 (Clustering Matrix)

- 把全域 A 矩陣分割成區域 (region) 時，依照 region 做 column/row shuffling，形成 C。
- $C = P * A * P^T$ ，是透過 permutation matrix P 做轉換。

簡單來說，C 就是根據目前分割後的新順序，重新排列的 A。

3. d_x / d_y : 目標向量 (Target Vector)

- 計算來自 固定 cell (pad) 的拉力。
- d_x 是針對 x 方向，d_y 是針對 y 方向。



舉例：假設 cell A 接了一個固定 cell pad，其 x 位置在 100，那麼 $d_x[A]$ 就會有一個偏向 100 的力。

最終要解的是這個最小化問題：

$$\text{Minimize: } x^T A x + 2d^T x \quad (\text{quadratic programming})$$

換句話說，要找到一組 x (cell 位置)，能讓線長 (由 A 表示) 與 pad 影響 (由 d 表示) 都最小。

◆詳細步驟：

step 1：建立 rowinfo_A 紀錄每區有哪些 cell

```
vector<MatrixInfo_Def> rowinfo_A;
rowinfo_A.resize(region_nbr);
for each cell:
    如果它是 movable (非固定 cell)
```

- 按照其所在區域加入對應的 list
- 並根據 cell 面積大小排序 (大的放前面)

rowinfo_A 是每一區的資訊集合。裡面記錄哪些 cell 要參與此輪的計算與排序順序。

step 2：建立打亂的矩陣 A、C、d 向量

for each region:

- 把這區的最大 cell 放在開頭 (大 cell 放第一欄)
- 把其他 cell 放在後面
- 同時建立 dx_cal / dy_cal 向量 (對應 d_x/d_y)
- 並打亂 C_matrix 的 row 形成 Cmatrix_tmp

打亂矩陣順序 (shuffle) 使得矩陣能夠對應每個區域的 cell 結構，準備進行矩陣運算。

step 3 : Cmatrix 的 column 也要 shuffle

for each region:

- 對應剛剛 row shuffle 的結果
- 做 column shuffle → 得到 Cmatrix_cal

step 4 : 矩陣運算解聯立方程式

```
mat x, y;
matrixCal(dx_cal, Cmatrix_cal, D, invD, B, Z, tranZ, &x, 0);
matrixCal(dy_cal, Cmatrix_cal, D, invD, B, Z, tranZ, &y, 1);
```

`matrixCal()` 是解出 $Ax + d = 0$ 的關鍵函數，這裡用的是一種叫 **spectral relaxation** 的方法來逼近最佳 cell 位置。根據公式：

$$\mathbf{x} = \mathbf{Z}(\mathbf{Z}^T \mathbf{C} \mathbf{Z})^{-1} (\mathbf{Z}^T (-\mathbf{C}\mathbf{x}_0 - \mathbf{d})) + \mathbf{x}_0$$

step 5 : 畫出目前結果

```
plotBMP(&fileId);
```

每一次迭代會畫一張 BMP 圖，顯示目前的 cell 分佈與分割狀況。

step 6 : 執行區域切割 partition()

```
partition_flag = partition(rowinfo_A, x, y);
```

根據目前算出的 cell 位置，嘗試進一步把一個區域分成兩半：

- 根據面積密度與比例計算合適的切割方向 (x/y)
- 如果該區域 cell 太少或面積太小，就不再切割

只要還能切割，`partition_flag` 會是正數，繼續迴圈；否則跳出。

◆總結

- **矩陣模型 (C, d_x, d_y)**：建立線性系統來計算最佳 cell 位置
- **Shuffling**：重新編排矩陣順序，對應分割區域
- **Partition**：根據 cell 密度與長寬比例動態切割空間
- **多次遞迴處理**：每次分割後再進行位置重算 → 越分越細 → 越精準

`int Gordian::partition(vector<MatrixInfo_Def> rowinfo_A, mat inputx, mat inputy)`

它是一個 **遞迴演算法**，會根據一個區域中的 cell 數量與範圍大小，**決定是否繼續將該區域一分為二**，同時調整 `rowinfo` 的 cell 分配。

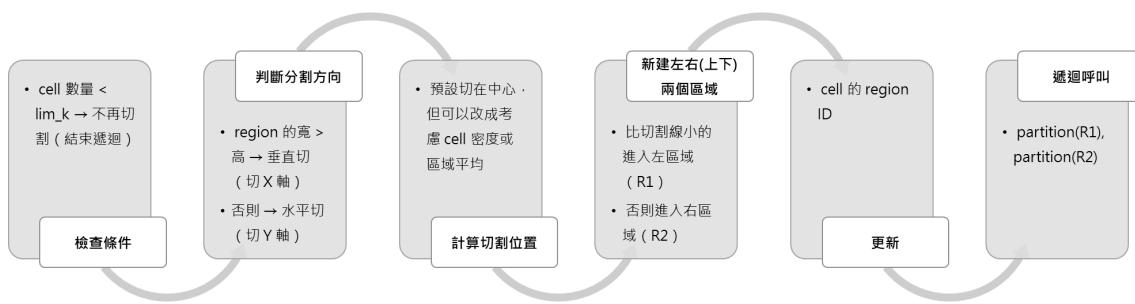
◆呼叫時機&參數說明：

在 `shuffle_()` 中，每次完成一次 matrix calculation 和位置更新後，`partition()` 就會檢查目前 cell 分布是否需要繼續切割：

```
if (partition(rowinfo[i], rowinfo, x_ans, y_ans))
    hasMorePart = true;
```

- `rowinfo_A`：每個 region 的 cell 資訊（哪些 cell 在這個 region、cell 數量等）。
- `inputx`, `inputy`：這些 cell 的 x 與 y 座標，來自外部計算（如 SVD 求解）。

◆ 流程圖：



◆詳細步驟：

step 1：將計算結果寫入 cell 座標

```
for (int i = 0; i < region_nbr; i++)
{
    if (i > 0)
        colpos += rowinfo_A.at(i - 1).cellnbr - 1;
    it = rowinfo_A.at(i).cellID_org.begin();
```

```

Cells[*it].cellpos[0] = inputx(i, 0);
Cells[*it].cellpos[1] = inputy(i, 0);
for (int j = 0; j < (rowinfo_A.at(i).cellnbr - 1); j++)
{
    it++;
    Cells[*it].cellpos[0] = inputx(j + colpos + region_nbr, 0);
    Cells[*it].cellpos[1] = inputy(j + colpos + region_nbr, 0);
}
}

```

目的：把 `inputx` 和 `inputy` (即 SVD/解線性方程式的結果) 寫進 `Cells[i].cellpos` 中，用於更新 cell 在空間中的位置。

index 計算說明：

- 第一個 cell 是 region 的代表點，它的座標在前面 (`inputx(i, 0)`)。
- 剩下的 cell 座標偏移 `colpos + region_nbr` 是因為原本的 A 矩陣有被重組成特殊形狀 (參考 `shuffle_` 生成的矩陣)。

step 2：找出需要切割的 region

```

parti_Step *= 2;
int reg, reg_left = 0;
for (reg = 0; reg < region_nbr; reg++)
{
    if (Region_Pos.cellsum.at(reg) <= lim_K) continue;
    ...
}

```

目的：檢查每個 region 的 cell 數是否超過限制 (`lim_K`)，若超過就進行切割。

step 3：決定切割方向

```

d_edgex = Region_Pos.edge_topX.at(reg) - Region_Pos.edge_botX.at(reg);
d_edgely = Region_Pos.edge_topY.at(reg) - Region_Pos.edge_botY.at(reg);
if (d_edgex >= d_edgely)
    Region_Pos.cutdir_flag.push_back(0); // x 方向
else
    Region_Pos.cutdir_flag.push_back(1); // y 方向

```

根據寬高比選擇切割方向：若 x 寬大 → 切 x (垂直切)、若 y 高大 → 切 y (水平切)

step 4：微調切割線 (迴圈)

```

while (1)
{
    ratio = ratioCal_2(...);
    if ((ratio < ratio_L) || (ratio > ratio_H))
    {
        // 根據 ratio 趨近 1 的方向來微調切割線
        // 使用 coe 累進收斂技巧
    }
}

```

```
...
}
else
{
    // 一旦 ratio 落在合理範圍，執行切割
    ...
    break;
}
}
```

- `ratio` 代表左右兩邊的面積比例（或 cell 數比例）
- 若比例失衡，就用調整係數 `coe` + direction sign 微調切割線
- 當比例合理，則：更新原本的 region 邊界、建立新 region 的邊界（Push_back）

step 5：更新區域數量與回傳切割數量

```
region_nbr = Region_Pos.region_cnt + 1;
return reg_left;
```

Conclusion

In this project, we successfully implemented and analyzed the **Gordian Placement algorithm**, a classical quadratic placement method for VLSI physical design. By transforming the global cell placement problem into a series of linear algebra problems and recursively applying region partitioning, the algorithm efficiently minimizes wirelength while achieving density balance.