

中原大學資訊工程學系

106 學年度專題

NET OPEN LOCATION

FINDER WITH OBSTACLES

指導教授：鄭維凱老師

組員：資工四乙 10327237 劉湘怡

資工四乙 10327250 陳佳琳

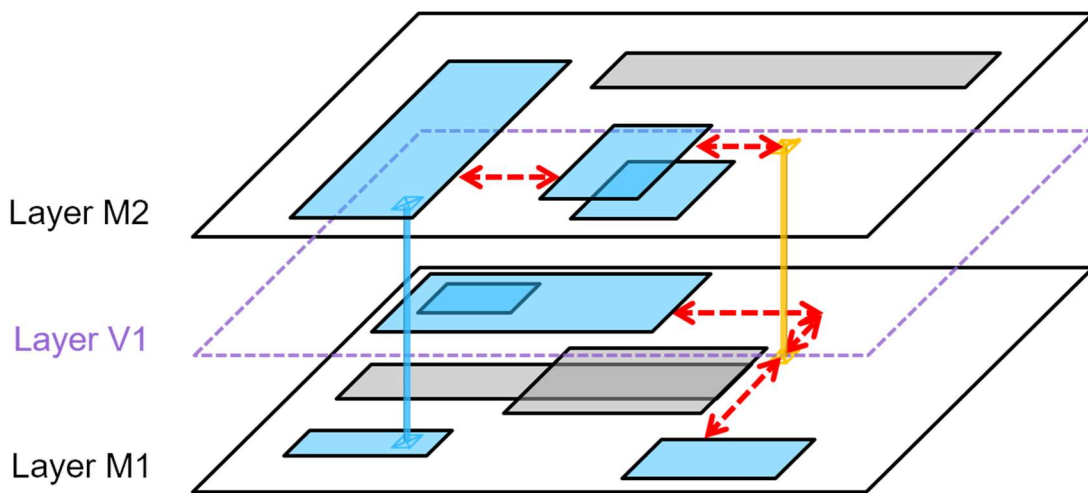
目錄

- 一、 專題簡介
- 二、 專題介紹
- 三、 過程構思
- 四、 流程圖
- 五、 專題運行結果
- 六、 結論
- 七、 心得
- 八、 附錄

一、專題簡介

起源：

在 IC 設計繞線（routing）時，為了節省成本及避免障礙物干擾，希望連接的線路能越短越好。以此為背景，此專題實作的過程我們討論了各種可能，運用資料結構及演算法，以及曾經學過的數學方法，達到線路之成本最小化。



- 此專題配合 2017 國際積體電路競賽 ICCAD 之 Problem B 做延伸。
- 將所有給定的藍色矩形相連接、並避免經過灰色矩形。
- 層與層間用給定的 via 或自訂 via 連接。
- 根據題目給的公式算出 **cost**，**cost** 越小分數越高。

二、專題介紹

如上圖，這是一個立體線路，每一層平面都可能有藍色矩形或灰色矩形。最終目的將所有題目給定的藍色矩形相連接成為一條條 path(如圖中紅色虛線)，並避免經過灰色矩形，而層與層之間須由 via 連接(如圖中黃色實線即為 via)。根據題目給的公式算出 cost(via 有 via 的 cost、path 則依其長度計算得其 path 之 cost)，得到的結果 cost 越小越好。

而我們在連接所有藍色矩形的過程中，會有很多問題是我們要去注意的：

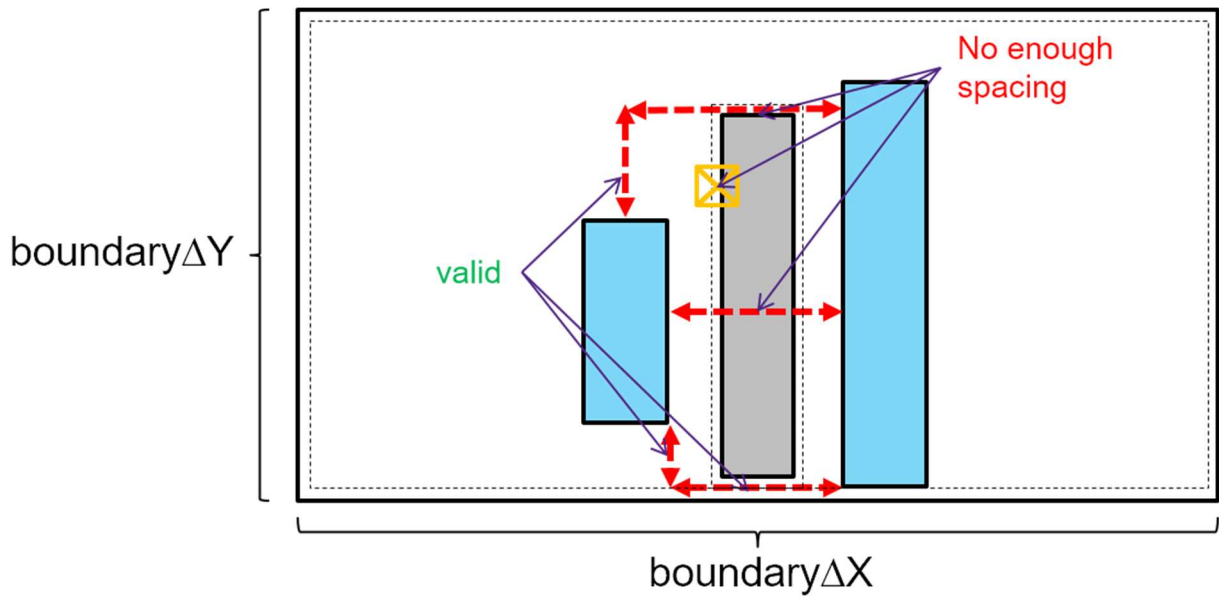
✚ Overall Cost = $\sum \text{Cost}(P_q)$

✚ 若 P_q 是 H-line 或 V-line， $\text{Cost}(P_q)$ 即為 H-line 或 V-line 的長度。

✚ 若 P_q 是 via， $\text{Cost}(P_q) = C_v$ 。

✚ 目標是將 overall cost 最小化

✚ 其中還有一些路徑是不合法(invalid)的，如：



- i. Path 沒離灰色矩形某個固定距離(依題目給定)
- ii. Path 不是水平線(H-line)或垂直線(V-line)
- iii. Paths are overlap.

註：

為了方便說明，

原題目之 **RoutedNetShape** 全文以藍色矩形稱之。

而原題目之 **Obstacle** 全文以灰色矩形稱之。

三、過程構思

矩形之間的連接

幾個月前第一次看到題目很茫然，不知道要怎麼把矩形們連接、卡了好多天，突然某天靈機一動想到學校老師曾教過的 Kruskal，若是我們能把矩形想像成點呢？那不就可以套 Kruskal 演算法下去做了嗎？也因為用到 Kruskal 演算法，我們才決定先做排序(才會因此用到 BubbleSort 跟 QuickSort)，且選擇用 BubbleSort 跟 QuickSort 這兩種，正是因為之後用的 Kruskal 演算法。

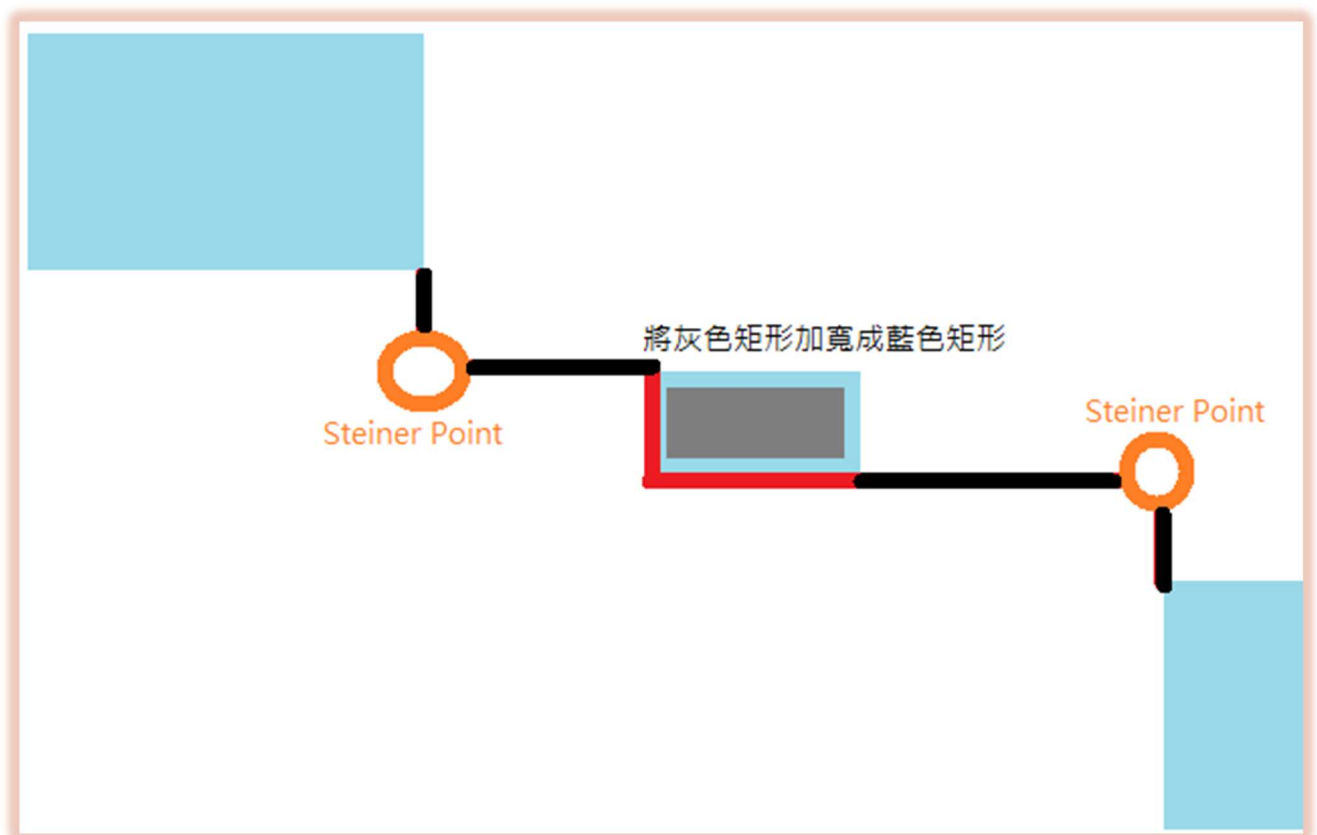
四分法

一開始用 Kruskal 演算法連接完整層的執行時間遠超乎我們想像，於是我們想有沒有方法能夠改善這種狀況，我們猜測執行時間跟我們丟進 Kruskal 演算法跑的矩形數量有關，那是不是改善數量就能解決呢？於是我們考慮幾種作法，主要是在四分法和十六分法之間做取捨。所謂的四分法是把每一層的矩形依相對位置分別存在四個象限，同理十六分法、六十四分法…等。其中我們發現，分存成愈多象限雖然大大減少執行時間，卻同時也增加了 path 之 cost，折衷之下，我們選擇用四分法。

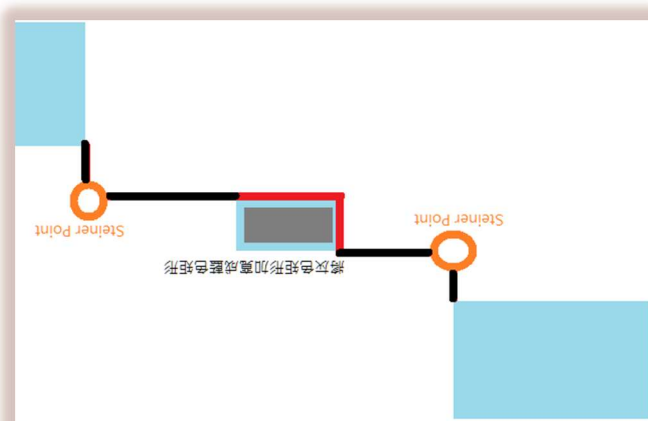
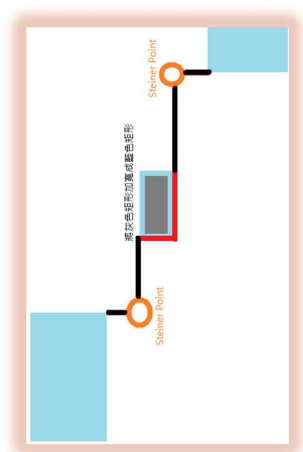
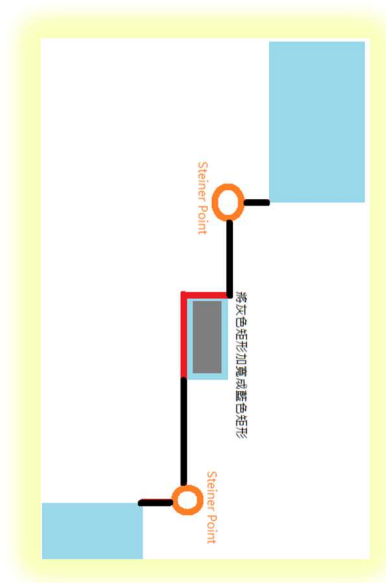
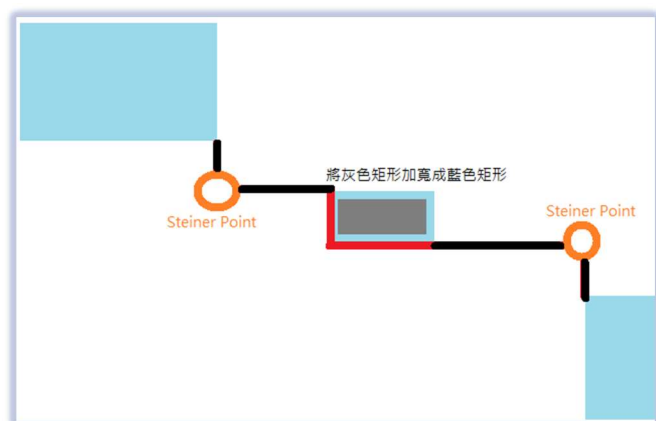
✚ 大野狼法(灰色矩形變藍色矩形)

據題目要求，連接藍色矩形的同時須避免經過灰色矩形，當時我們想了四天，要讓程式自動偵測相對位置去避開灰色矩形，不易做到且耗時耗成本，因此我們想了個方法，像童話故事<<小紅帽與大野狼>>中的大野狼假扮奶奶，要是我們把灰色矩形假扮成藍色矩形呢？因為我們這樣的做法，成功地簡單避開灰色矩形。

但須特別注意的是，那些曾經假扮成藍色矩形的灰色矩形，連接完後要記得加上 L 型 path，才算是有連接上。(如下圖紅色實線所示)



其中，L 型可以 360 度旋轉如下(注意圖中兩藍色矩形間相對位置)：

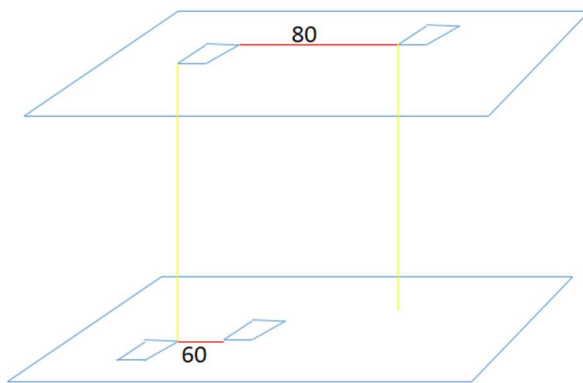


投影法

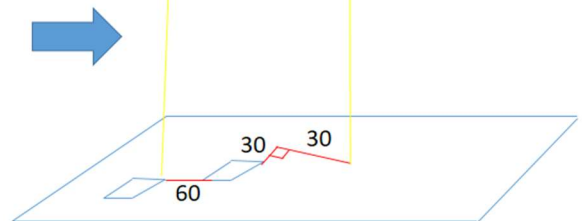
在決定做投影法的前一天，我們因為結果不理想(cost 太大)，當時我們用的每一層矩形先相互連接，再用少量的 via 做層與層之間的連接，發現這樣做 cost 太大，因此有了投影法的發想。

投影法顧名思義，把上層的矩形通通投影至下層，以布林變數判斷上下層矩形，其中上層與下層的連接要算上 via 之 cost。再以 Kruskal 演算法作連接。

註:黃線為via，此例via之cost為10

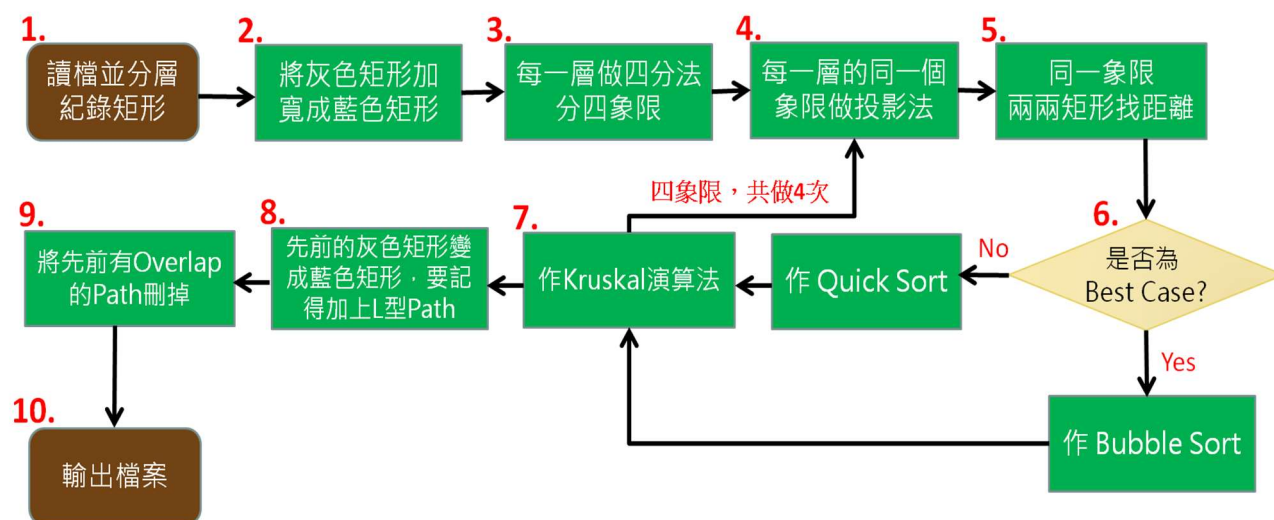


使用最初方法得Overall cost = $60+80+10*1=150$



使用投影法後，圖中4個藍色矩形兩兩相連，用Kruskal演算法，得需用得最短三條線分別為10、60、70($30+30+10$)，所以總cost= $10+60+70=140$

四、 流程圖

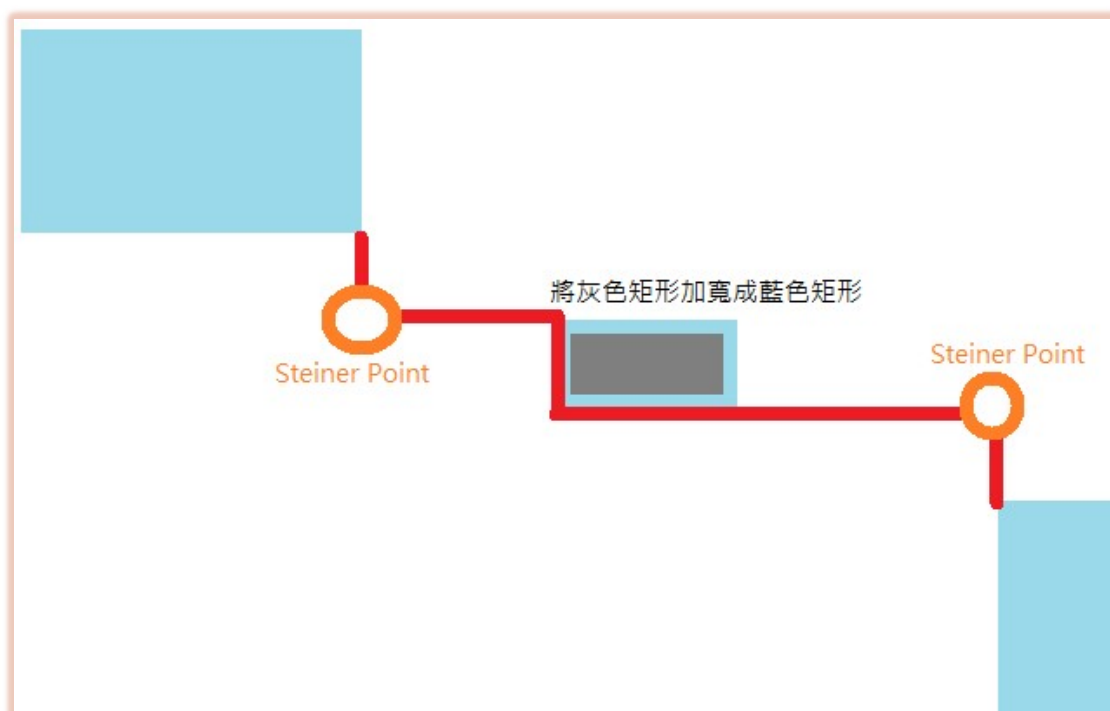


1. 讀檔並分層記錄矩形：藍色矩形及灰色矩形

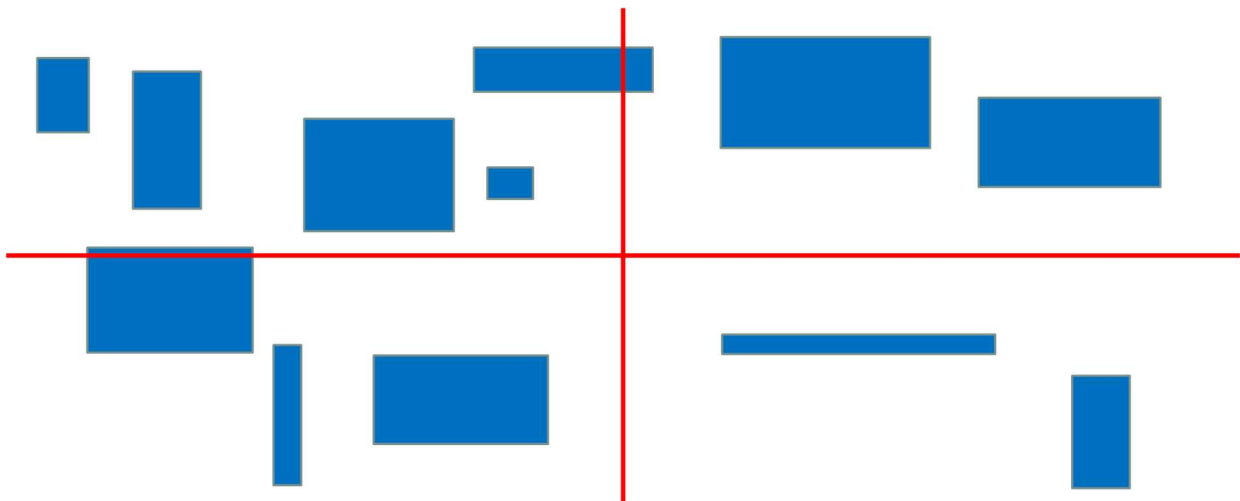
2. 大野狼法：把灰色矩形加寬變成藍色矩形

(此方法可避免 path 穿過灰色矩形，形成不合法的路徑)

如下圖所示：

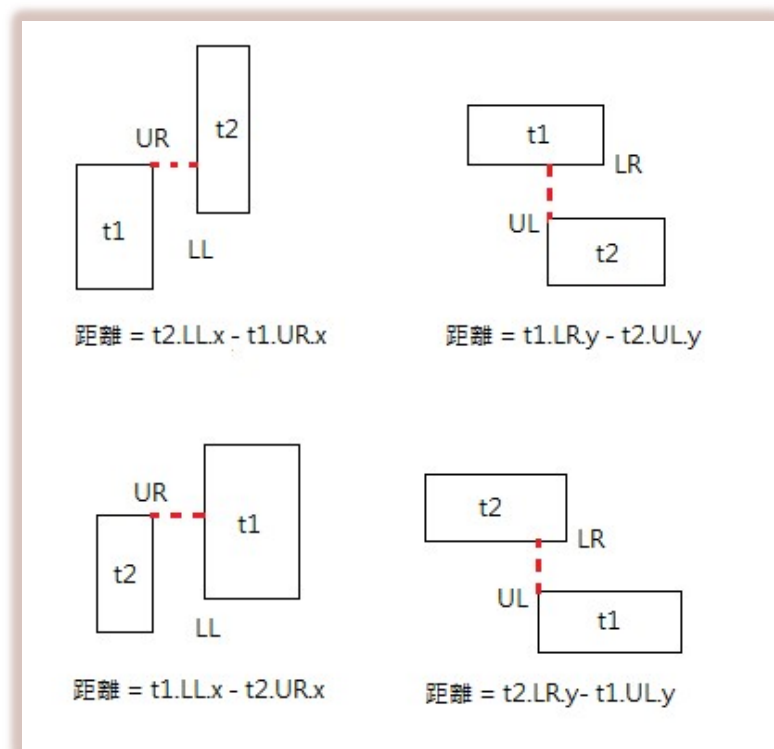


3. 每一層平面將藍色矩形做四分法分四象限，四分法如下圖所示

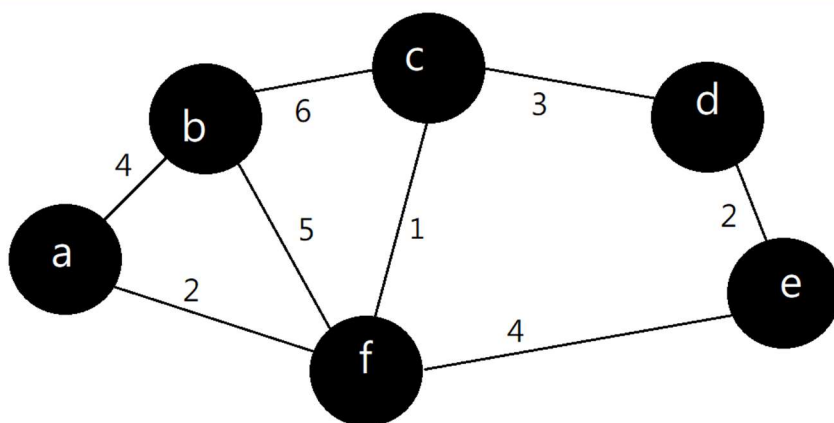


4. 針對某一象限作投影法(統一上層投影至下層、連接 via)

5. 同一象限兩兩藍色矩形做連接，計算並記錄其兩兩距離，如下圖所示(紅色虛線即為兩兩矩形之間的最短距離)。



6. 依 Best Case 與否決定用 BubbleSort 還是 QuickSort 作排序，將藍色矩形間兩兩距離由小排到大。
7. 排序完後，用 Kruskal 演算法保證連接圖上同一象限每一矩形。如下圖，假設集合 $S=\{a,b,c,d,e,f\}$ (其中 $a\sim f$ 皆為矩形)， A 是空集合，利用 Kruskal 演算法，先將 c 、 f 加入集合 A (\overline{cf} 最短)，另外記錄其 path，加入集合 A 後，集合 S 刪掉 c 、 f ， $S=\{a,b,d,e\}$ ，再依序連接 \overline{af} 、 \overline{de} 、 \overline{cd} 、 \overline{ab} 、 \overline{ef} 、 \overline{bf} 、 \overline{bc} 以及刪掉 S 目前有的矩形： a 、 d 、 e 、 b ，直至集合 S 為空，表示全部矩形連接完成。



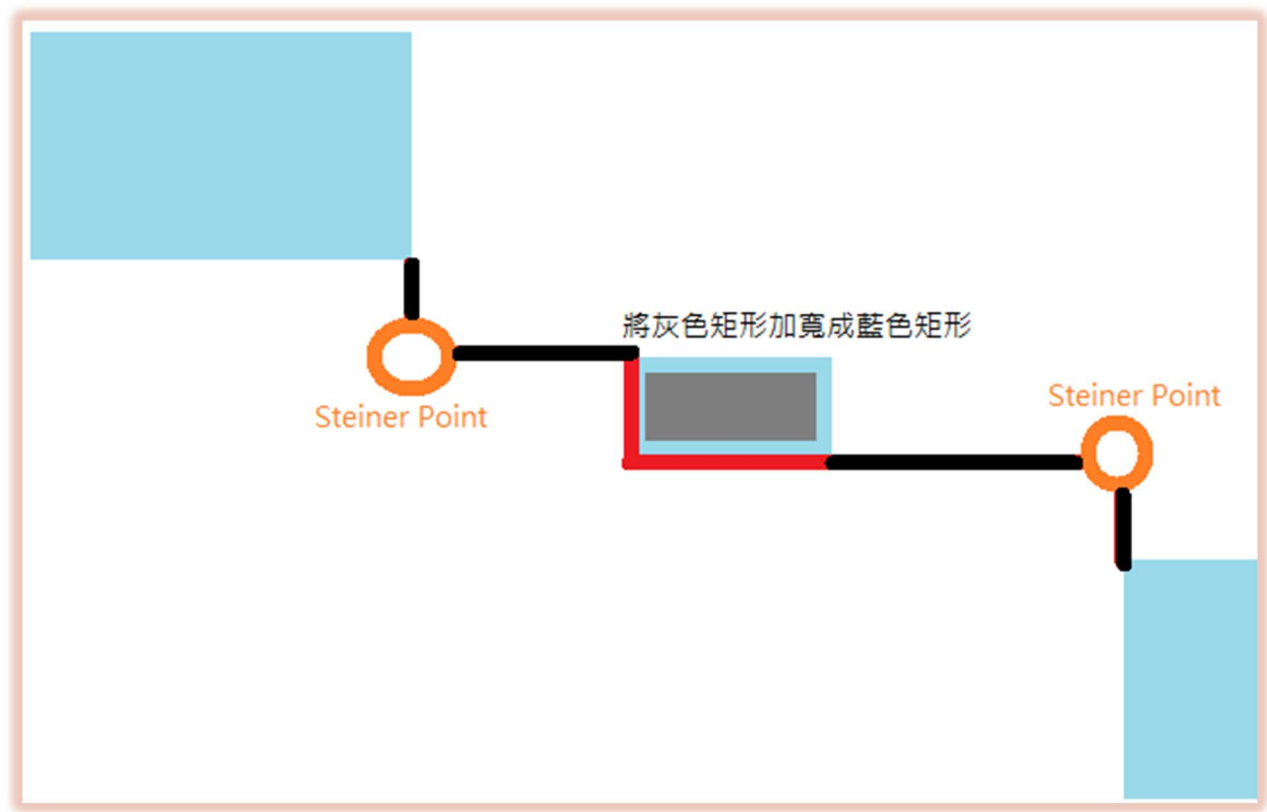
Kruskal's Algorithm

```

A={}, Kruskal(V,E)
A=∅
foreach v ∈ V:
    Make-disjoint-set(v)
Sort E by weight increasingly
foreach (v1,v2) ∈ E:
    if Find(v1)≠Find(v2):
        A=A∪{(v1,v2)}
        Union (v1,v2)
return A

```

8. 把假藍色矩形(亦即先前的大野狼灰色矩形)，加上 L 型 path



(重複第四步驟到第八步驟，四象限共做四次)

9. 把先前有 overlap 的線刪掉(包含部分 overlap 或者全部 overlap 的 path 們)

10. 輸出檔案

五、專題運行結果

case1 之總 cost 為 4776(path 之 cost 為 4326，via 之 cost 為 450)，跑了 94 秒。

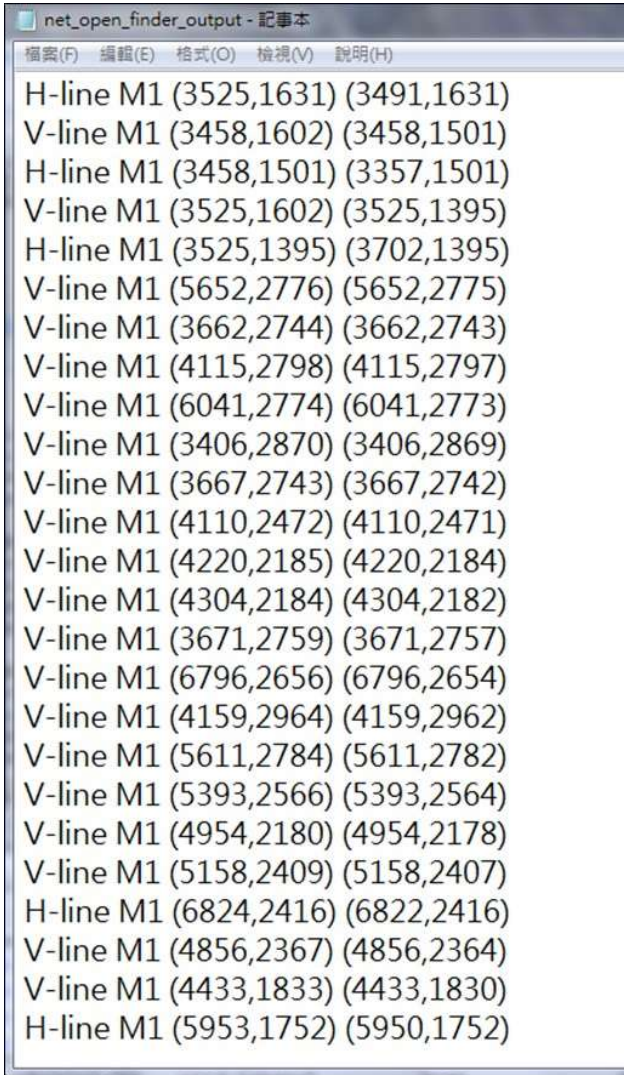
case2 之總 cost 為 45386(path 之 cost 為 39806，via 之 cost 為 5580)，跑了 1019

秒。 case5 之總 cost 為 18196(path 之 cost 為 13691，via 之 cost 為 4505)，跑

了 939 秒。

(以上數據在 WINDOWS 10 下、INTEL® CORE™ i5-4210U (ASUS X555L) 下執行)

附圖為擷取匯出檔之部分 V-line、H-line：



```
net_open_finder_output - 記事本
檔案(F) 編輯(E) 格式(O) 檢視(V) 說明(H)
H-line M1 (3525,1631) (3491,1631)
V-line M1 (3458,1602) (3458,1501)
H-line M1 (3458,1501) (3357,1501)
V-line M1 (3525,1602) (3525,1395)
H-line M1 (3525,1395) (3702,1395)
V-line M1 (5652,2776) (5652,2775)
V-line M1 (3662,2744) (3662,2743)
V-line M1 (4115,2798) (4115,2797)
V-line M1 (6041,2774) (6041,2773)
V-line M1 (3406,2870) (3406,2869)
V-line M1 (3667,2743) (3667,2742)
V-line M1 (4110,2472) (4110,2471)
V-line M1 (4220,2185) (4220,2184)
V-line M1 (4304,2184) (4304,2182)
V-line M1 (3671,2759) (3671,2757)
V-line M1 (6796,2656) (6796,2654)
V-line M1 (4159,2964) (4159,2962)
V-line M1 (5611,2784) (5611,2782)
V-line M1 (5393,2566) (5393,2564)
V-line M1 (4954,2180) (4954,2178)
V-line M1 (5158,2409) (5158,2407)
H-line M1 (6824,2416) (6822,2416)
V-line M1 (4856,2367) (4856,2364)
V-line M1 (4433,1833) (4433,1830)
H-line M1 (5953,1752) (5950,1752)
```

六、結論

Case1 專題運行結果比較

版本	層數	藍色 矩形 個數	灰色 矩形 個數	總 Path	共使用 via 之 數目	總 viaCost (此 case 都是 10)	總 cost= 總 Path+ 總 viaCost	執行所 需時間
最初	3	1503	414	53095736	2	20	53095756	204 秒
最終				4326	45	450	4776	94 秒

Case2 專題運行結果比較

版本	層數	藍色 矩形 個數	灰色 矩形 個數	總 Path	共使用 via 之 數目	總 viaCost (此 case 都是 30)	總 cost= 總 Path+ 總 viaCost	執行所需時間
最初	5	4518	4773	515899298	4	120	515899418	18669 秒 (約 5.1 小時)
最終				39806	186	5580	45386	1019 秒

Case5 專題運行結果比較

版本	層數	藍色 矩形 個數	灰色 矩形 個數	總 Path	共使用 via 之 數目	總 viaCost (此 case 都是 5)	總 cost= 總 Path+ 總 viaCost	執行所需時間
最初	5	4450	4762		4	20		
最終				13691	901	4505	18196	939 秒

最初版本是尚未用投影法時得到的結果，此時我們層與層之間只用一條 via。

從數據上看，我們可以發現用了投影法後，不只 cost 大幅減少、執行時間也大幅降低。

結論：(在 via 之 cost 小於 100、layer 層數超過三層的情況下)


雖然使用大量 via 造成 via 之 cost 增加，但整體 cost 比起層層之間只有一條 via 來的好。

七、心得

這幾個月下來，我們從剛開始的每周一聚到後來每天從早到晚都在語音討論專題、甚至有四天連假都待在學校討論(那時候還沒想法、想法設法就是沒想到怎麼把架構接起來)，從一開始的六七百行架構到後來的兩三千行程式碼，不斷嘗試其他新方法以減少我們的 `cost` 和縮短執行時間，我們歷經 `Beta test` 前夕連續幾天不睡覺、補習班翹課只為了優化我們的 `code`，最終我們做出來了!從 8/17 case1 的 `cost` 大小八位數到最終的四位數，真的很令我們感動!青春不虛此行!

八、附錄


專題過程

 2017/7/31

1. 把灰色矩形假扮成藍色矩形(大野狼法)
2. 用四分法把每一層分成四個象限
3. 用 Bubble Sort 先把兩兩矩形之間的距離由小排到大(方便後面 Kruskal 之用)
4. 每一層每一象限的藍色矩形相互連接(Kruskal 演算法)
5. 每一層的四個象限彼此連接(360 度 Γ 字型)
6. 層與層間只用一個 via 連接


此時測得 case1 跑了 1161 秒，而 case2 跑不出來(太久)，

這是我們第一次完整連結所有矩形…仍須努力!

 2017/8/3


發現 7/31 的 bug：假矩形並沒有真正連接，至少還差水平線與垂直線各一條 (L 型 path)，改善 bug 並成功縮短執行時間。

此時 case1 跑了 371 秒，case2 跑了 33354 秒(9.265hr)。


 2017/8/7

判斷是否為 Best Case 決定採用 BubbleSort 或 QuickSort，


此時 case1 跑了 245 秒，而 case2 跑了 24495 秒(6.8hr)。

 2017/8/9


第一次測試我們的 output 檔的 cost，光是 case1 之 cost 就有 98147671，與我們預期落差太大，經過我們仔細審視我們的程式碼、決定大改!

 2017/8/14

刪掉大於 Viacost 的 path，此時 case1 之 cost 為 53095736 跑了 204 秒，仍是八位數且已經刪了大於 ViaCost 的線...我們很灰心但永不放棄!

 2017/8/19

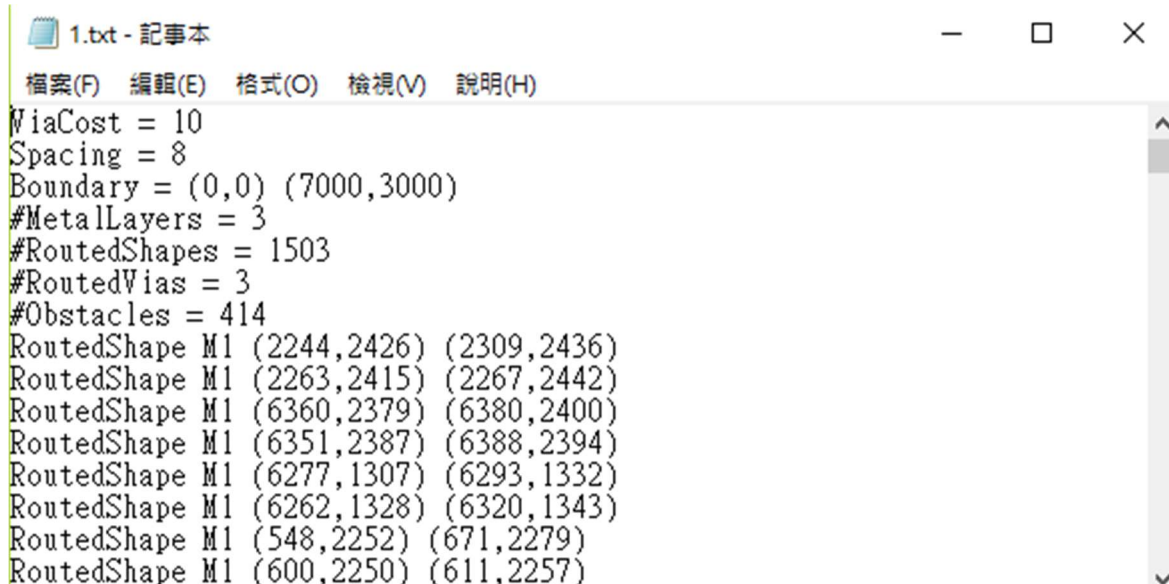
改了 Erase() 刪 RNS_2 直接用 RNS_4，不把灰色矩形假扮成藍色矩形，最後發現還是使用大野狼法效果比較好~此時 case1 之 cost 為 77542(大進步!)，跑了 173 秒；而 case2 之 cost 為 45780，跑了 837 秒。

 2017/9/8(最終版本)

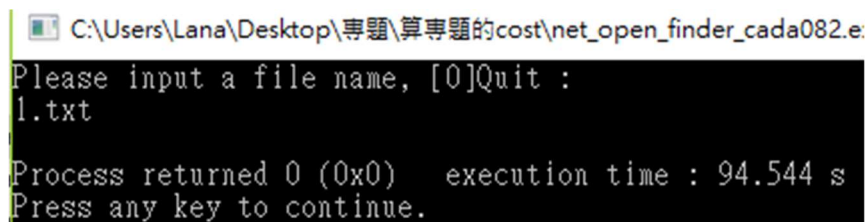
使用投影法、將立體圖形平面化--用多點 Via(Via 之 cost 增加)來減少總 Cost，此時 case1 之 cost 為 4776，跑了 94 秒；而 case2 之 cost 為 45386，跑了 1019 秒。

運行結果附圖

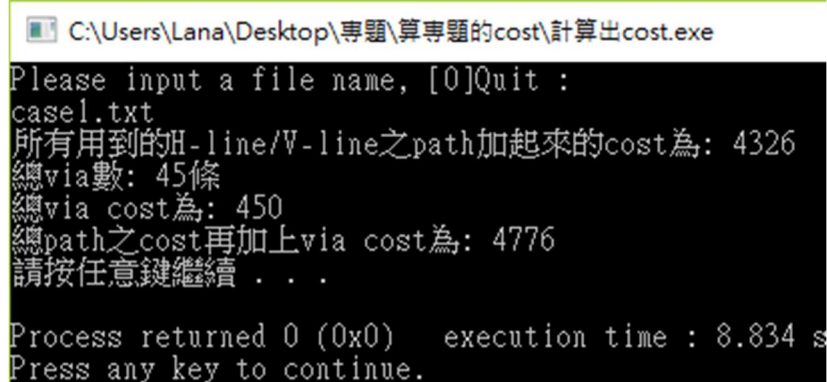
Case1



```
1.txt - 記事本
檔案(F) 編輯(E) 格式(O) 檢視(V) 說明(H)
ViaCost = 10
Spacing = 8
Boundary = (0,0) (7000,3000)
#MetalLayers = 3
#RoutedShapes = 1503
#RoutedVias = 3
#Obstacles = 414
RoutedShape M1 (2244,2426) (2309,2436)
RoutedShape M1 (2263,2415) (2267,2442)
RoutedShape M1 (6360,2379) (6380,2400)
RoutedShape M1 (6351,2387) (6388,2394)
RoutedShape M1 (6277,1307) (6293,1332)
RoutedShape M1 (6262,1328) (6320,1343)
RoutedShape M1 (548,2252) (671,2279)
RoutedShape M1 (600,2250) (611,2257)
```

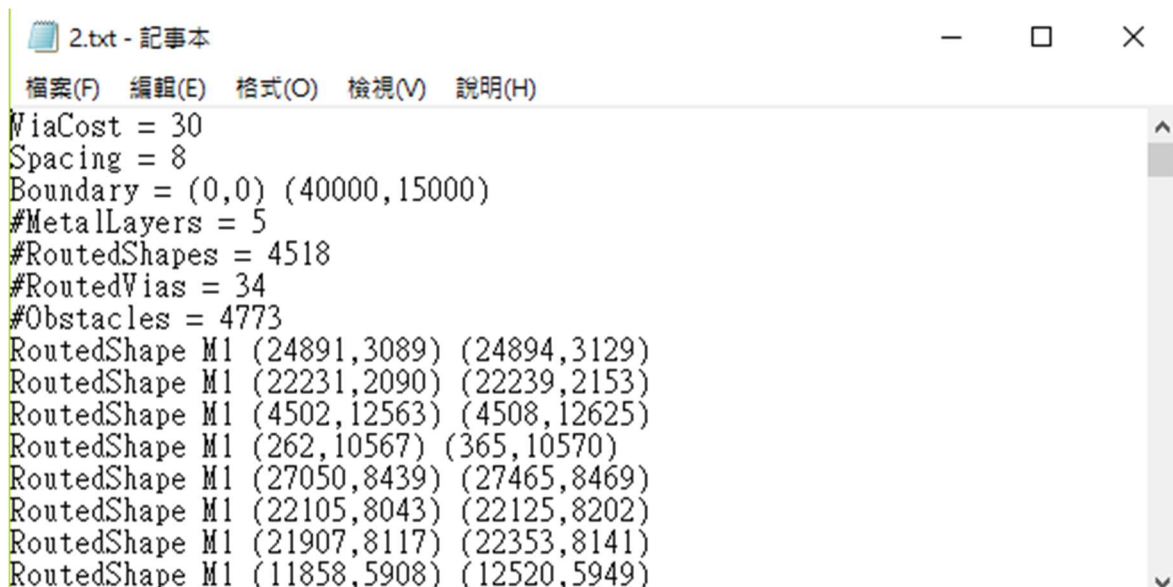


```
C:\Users\Lana\Desktop\專題\算專題的cost\net_open_finder_cada082.e
Please input a file name, [0]Quit :
l.txt
Process returned 0 (0x0)   execution time : 94.544 s
Press any key to continue.
```

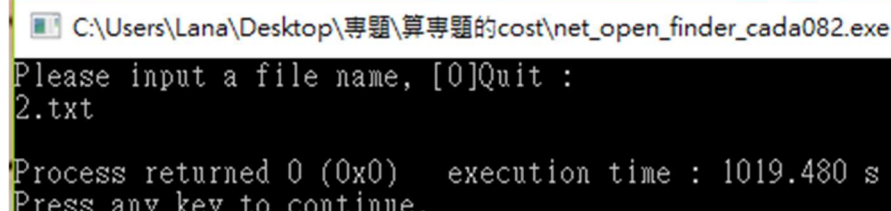


```
C:\Users\Lana\Desktop\專題\算專題的cost\計算出cost.exe
Please input a file name, [0]Quit :
casel.txt
所有用到的H-line/V-line之path加起來的cost為: 4326
總via數: 45條
總via cost為: 450
總path之cost再加上via cost為: 4776
請按任意鍵繼續 . . .
Process returned 0 (0x0)   execution time : 8.834 s
Press any key to continue.
```

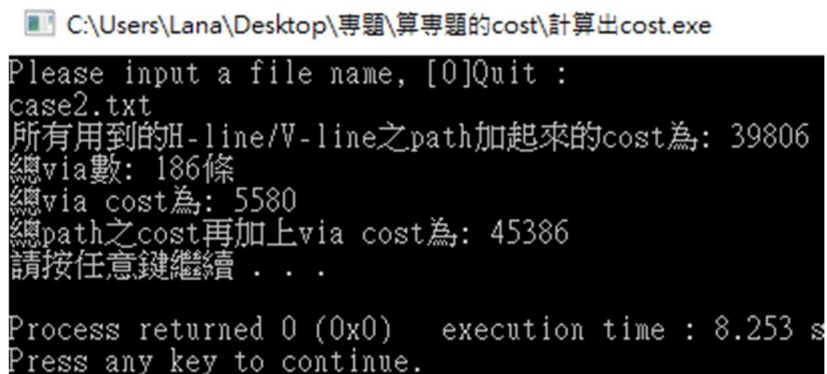
Case2



```
2.txt - 記事本
檔案(F) 編輯(E) 格式(O) 檢視(V) 說明(H)
ViaCost = 30
Spacing = 8
Boundary = (0,0) (40000,15000)
#MetalLayers = 5
#RoutedShapes = 4518
#RoutedVias = 34
#Obstacles = 4773
RoutedShape M1 (24891,3089) (24894,3129)
RoutedShape M1 (22231,2090) (22239,2153)
RoutedShape M1 (4502,12563) (4508,12625)
RoutedShape M1 (262,10567) (365,10570)
RoutedShape M1 (27050,8439) (27465,8469)
RoutedShape M1 (22105,8043) (22125,8202)
RoutedShape M1 (21907,8117) (22353,8141)
RoutedShape M1 (11858,5908) (12520,5949)
```

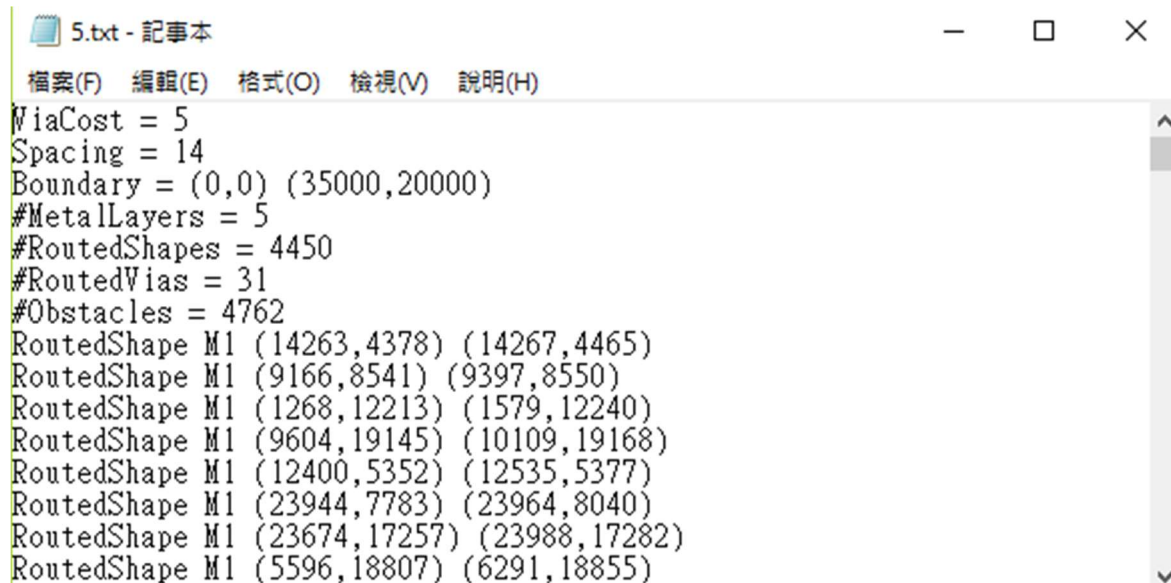


```
C:\Users\Lana\Desktop\專題\算專題的cost\net_open_finder_cada082.exe
Please input a file name, [0]Quit :
2.txt
Process returned 0 (0x0)   execution time : 1019.480 s
Press any key to continue.
```

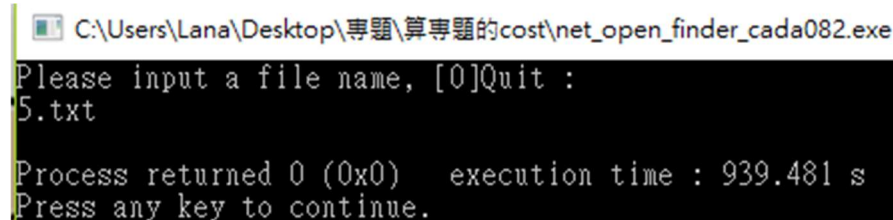


```
C:\Users\Lana\Desktop\專題\算專題的cost\計算出cost.exe
Please input a file name, [0]Quit :
case2.txt
所有用到的H-line/V-line之path加起來的cost為: 39806
總via數: 186條
總via cost為: 5580
總path之cost再加上via cost為: 45386
請按任意鍵繼續 . . .
Process returned 0 (0x0)   execution time : 8.253 s
Press any key to continue.
```

Case5(檔名叫 case5.txt)



```
5.txt - 記事本
檔案(F) 編輯(E) 格式(O) 檢視(V) 說明(H)
ViaCost = 5
Spacing = 14
Boundary = (0,0) (35000,20000)
#MetalLayers = 5
#RoutedShapes = 4450
#RoutedVias = 31
#Obstacles = 4762
RoutedShape M1 (14263,4378) (14267,4465)
RoutedShape M1 (9166,8541) (9397,8550)
RoutedShape M1 (1268,12213) (1579,12240)
RoutedShape M1 (9604,19145) (10109,19168)
RoutedShape M1 (12400,5352) (12535,5377)
RoutedShape M1 (23944,7783) (23964,8040)
RoutedShape M1 (23674,17257) (23988,17282)
RoutedShape M1 (5596,18807) (6291,18855)
```



```
C:\Users\Lana\Desktop\專題\算專題的cost\net_open_finder_cada082.exe
Please input a file name, [0]Quit :
5.txt

Process returned 0 (0x0)   execution time : 939.481 s
Press any key to continue.
```



```
C:\Users\Lana\Desktop\專題\算專題的cost\計算出cost.exe
Please input a file name, [0]Quit :
case5.txt
所有用到的H-line/V-line之path加起來的cost為: 13691
總via數: 901條
總via cost為: 4505
總path之cost再加上via cost為: 18196
```

參考資料

 [ICCAD 2017 國際積體電路電腦輔助設計軟體製作競賽](#)

http://cad-contest-2017.e1.cycu.edu.tw/Problem_B/default.html(本文第三頁的圖來自於此)

 [Wikipedia Kruskal's Algorithm](#)

https://en.wikipedia.org/wiki/Kruskal%27s_algorithm

 [Algorithms and Data Structures](#)

http://www.algolist.net/Algorithms/Sorting/Bubble_sort

 [GeeksforGeeks](#)

<http://www.geeksforgeeks.org/quick-sort/>