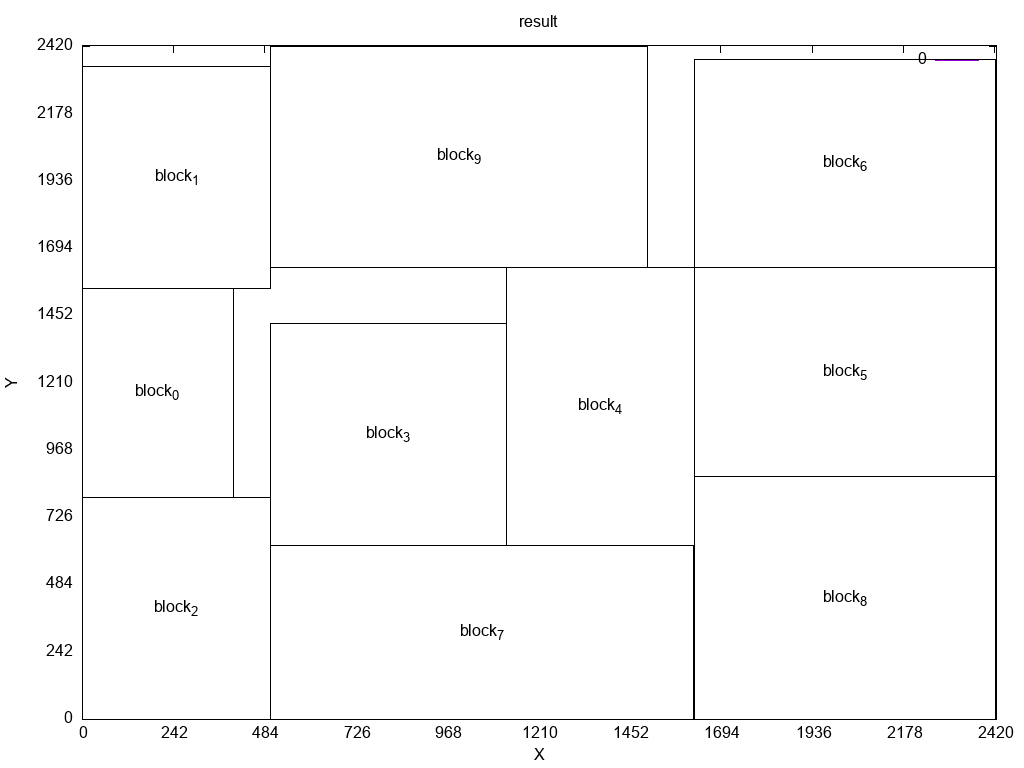
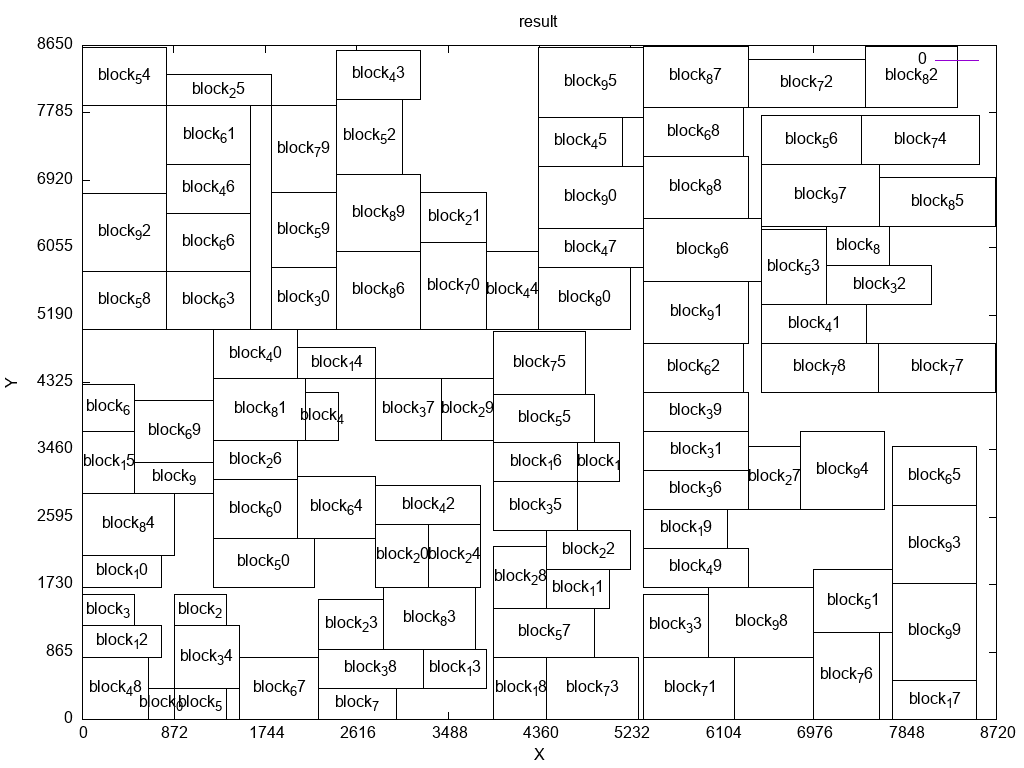
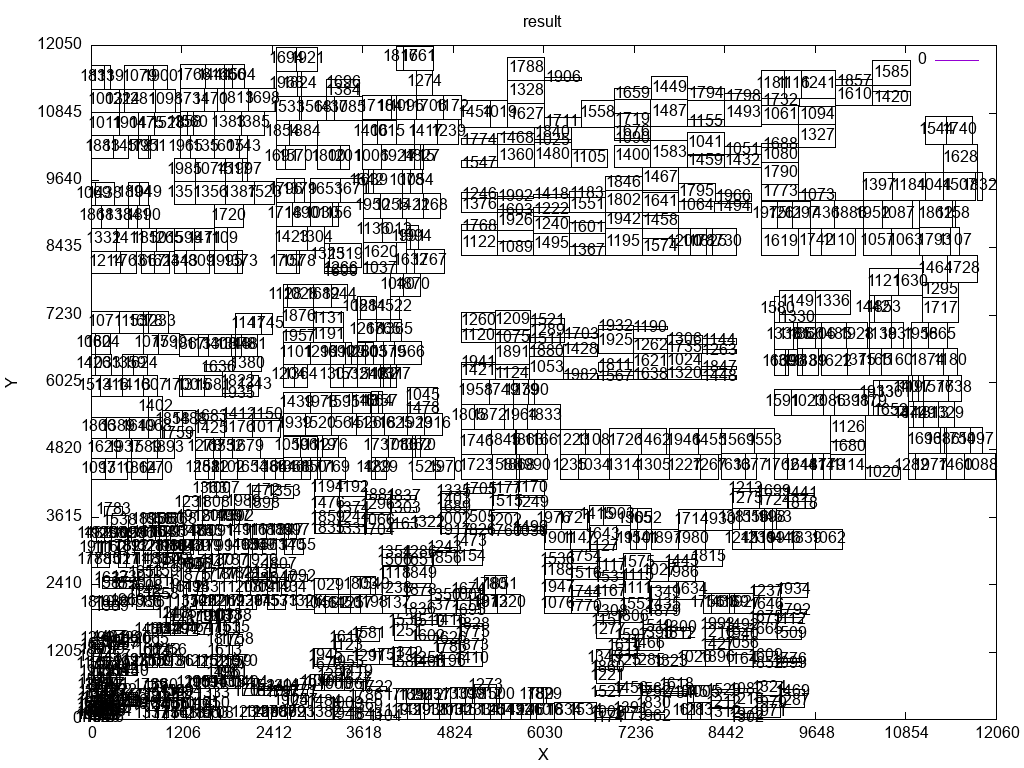
**Software Development for Electronic Design Automation**

**Spring 2024**

**Programming Assignment #2 Soft Block Floorplanning**

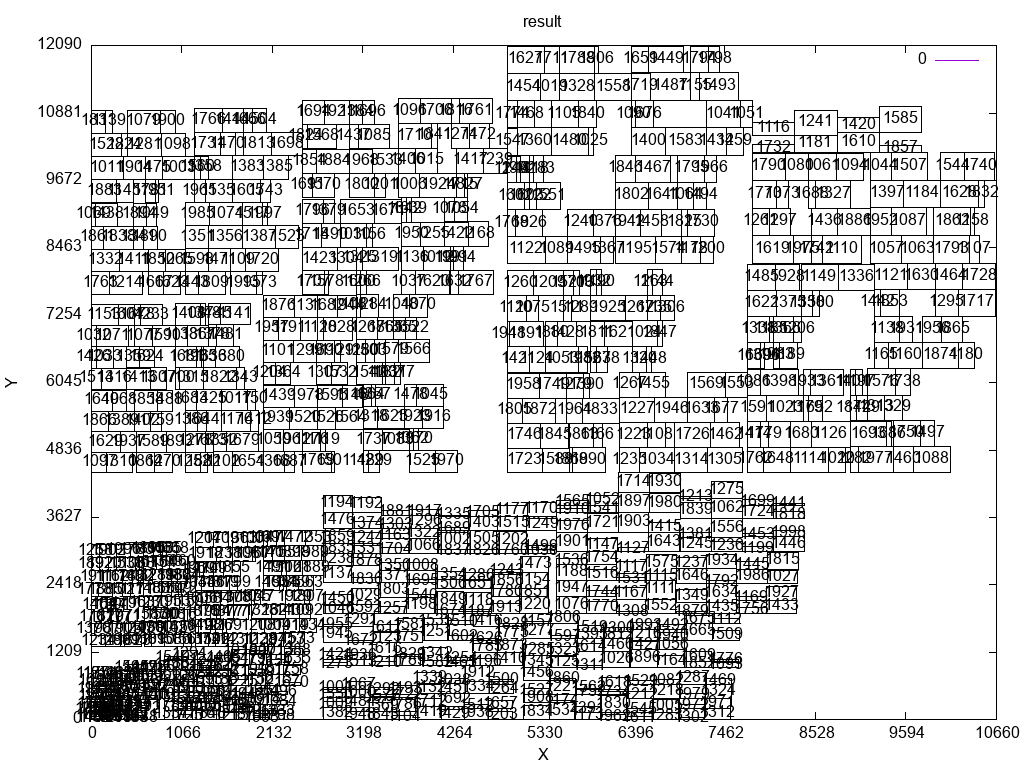
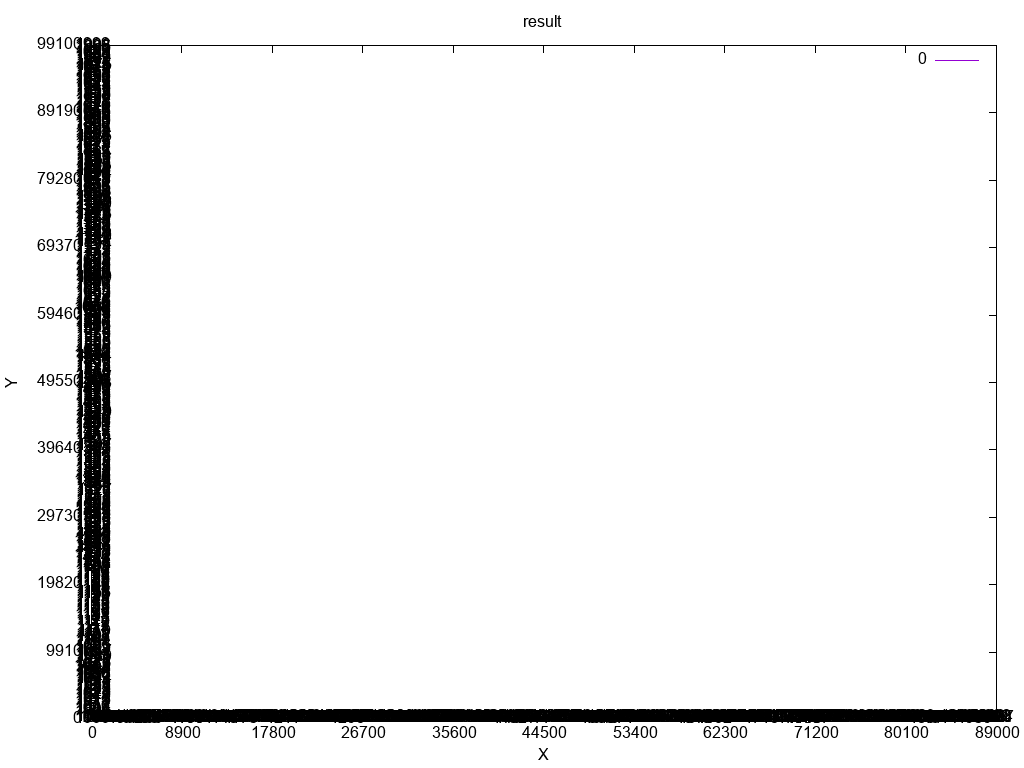
**B10902227 吳宗喬 四電子四丙**

　　本次作業我選擇slicingtree搭配SA為基礎，加入些額外演算法來實現，主要的資料結構有block與slicingtree，採用合併式演算法初始化slicingtree，並在初始化後使用SA演算法找出優質解。Block包含元件放置所需的基本資料，如寬、高、平面位置與其名稱。Slicingtree包含slicingtree的representation sequence，映射的blocks與加速M3操作檢測的checking sequence。初始化的演算法會將相近邊長的bloks倆倆合併，形成大bloks，再將有相近邊長的大bloks倆倆合併，形成巨大blocks，如此下去，直到所有blocks被合併成一個block，此合併過程生成的sequence作為representation sequence的初始值，這種方式保證初始分布是平均且有根據的(非隨便亂放的)。SA會配合課上所講的三個representation sequence操作進行更新，但要注意其cost數字極大，因此溫度的scale要非常大(我由1000.0開始)，才有正常的機率去更新。整個程式可以分為三個步驟，第一步會讀取input file並根據輸入的blocks去初始化slicingtree，第二步會對slicingtree進行SA去找優質解，第三步slicingtree會根據representation sequence將其映射的blocks進行定位(放置)並將擺放的結果輸出成.out格式。

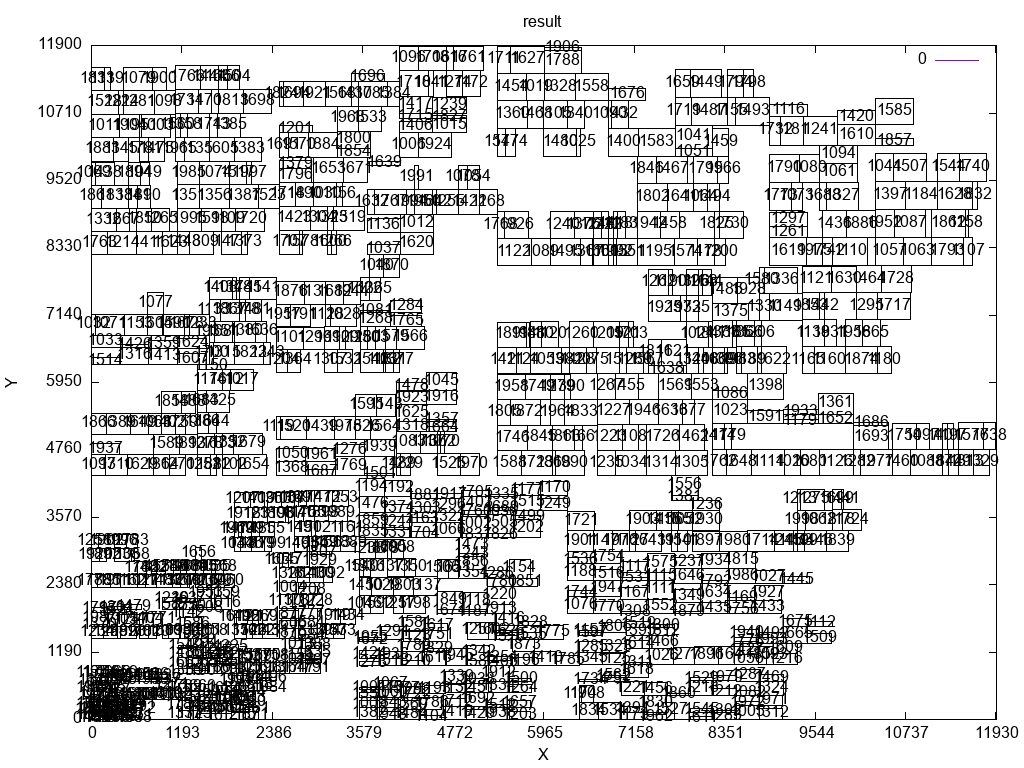
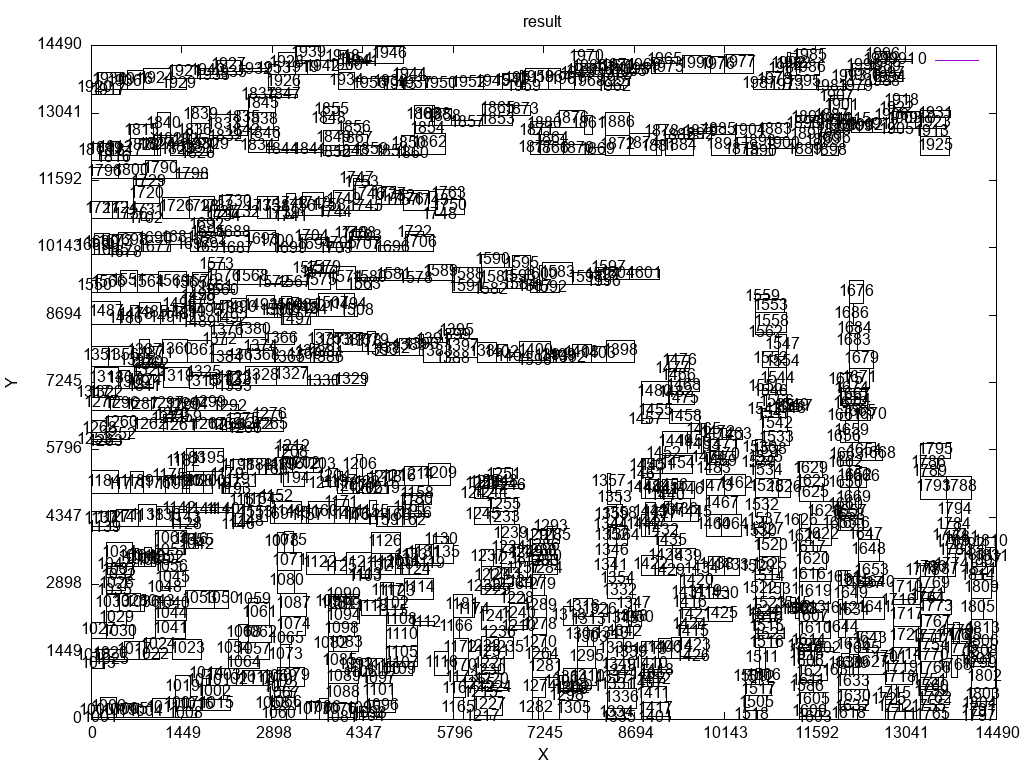
  

**圖(1)、左至右分別為floorplan 10, 100, 1000下的結果。**

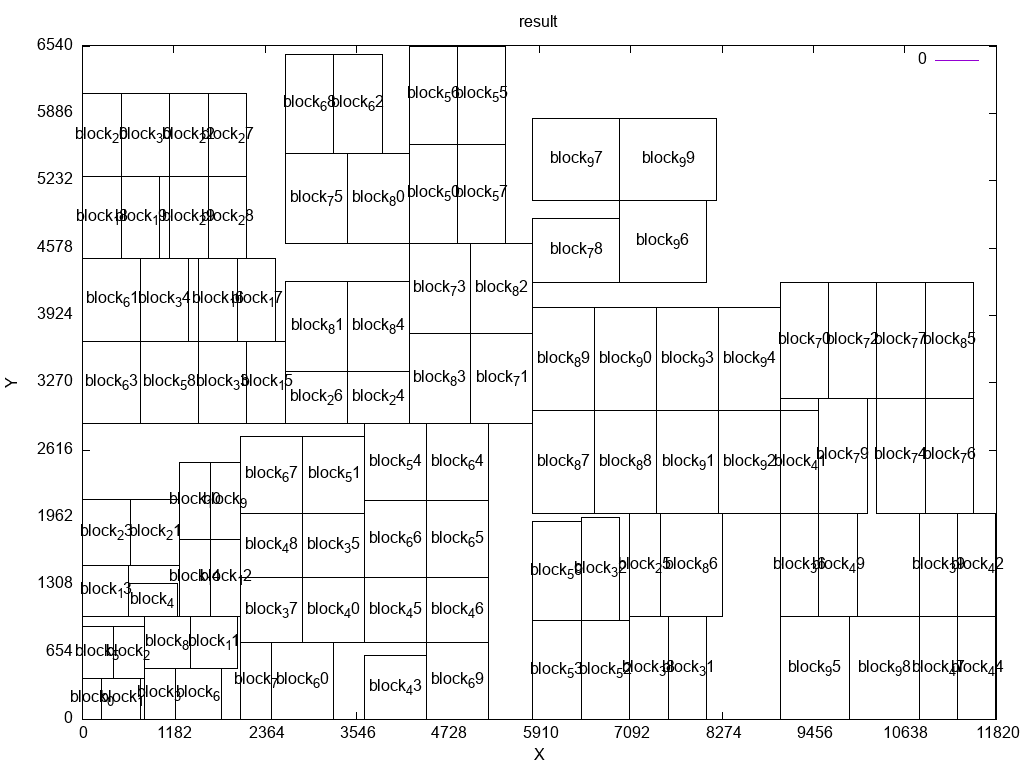
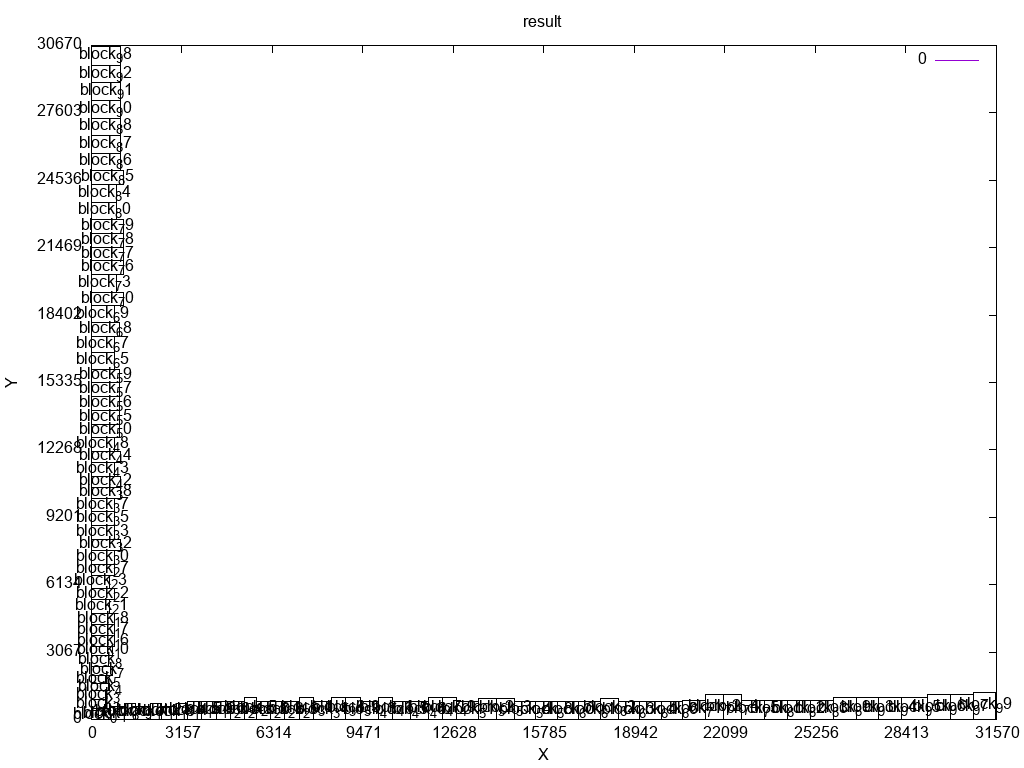
在slicingtree的初始化上，我原先是簡單的整排放置，如圖(2)(4)的左圖，但當SA的移動次數不足時，會導致沒移動的representation sequence整排保留到結果，形成大量串列並留下大量空洞，如圖(3)左圖可以明顯看出在blocks數量過大時，會有很長一串的blocks影響優化結果，因此，我改用合並演算法進行初始化，得到圖(3)右圖的優化結果，使我的SA的次數不必隨block量級急遽成長，因為它良好的初始狀態可以更好去尋找解，這些都是為了應付大量blocks的演算法，當blocks量級不大時，其結果差不多，比較圖(3)與圖(5)，初始化演算法的在blocks在1000與100量級下的影響性，明顯在量級1000的影響更大，再看看圖(6)，當blocks的量級到5000時，合併初始化的結果比整排放置的好29倍(看Optimal Cost)，在量級提升下對結果的差距非常可觀。



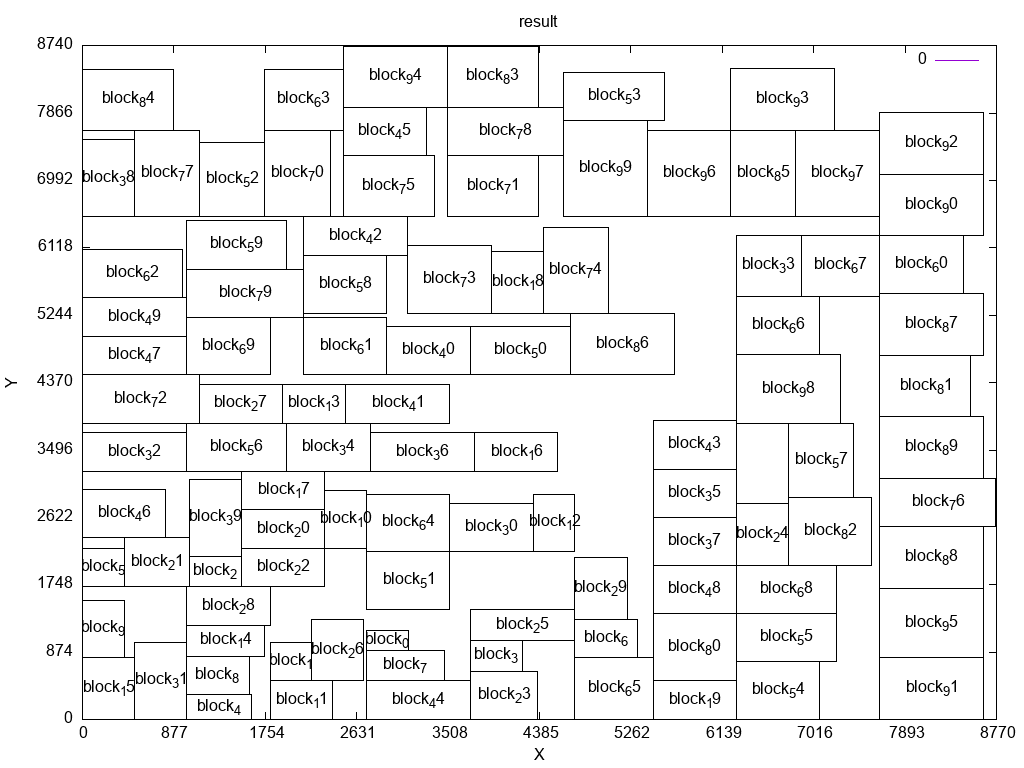
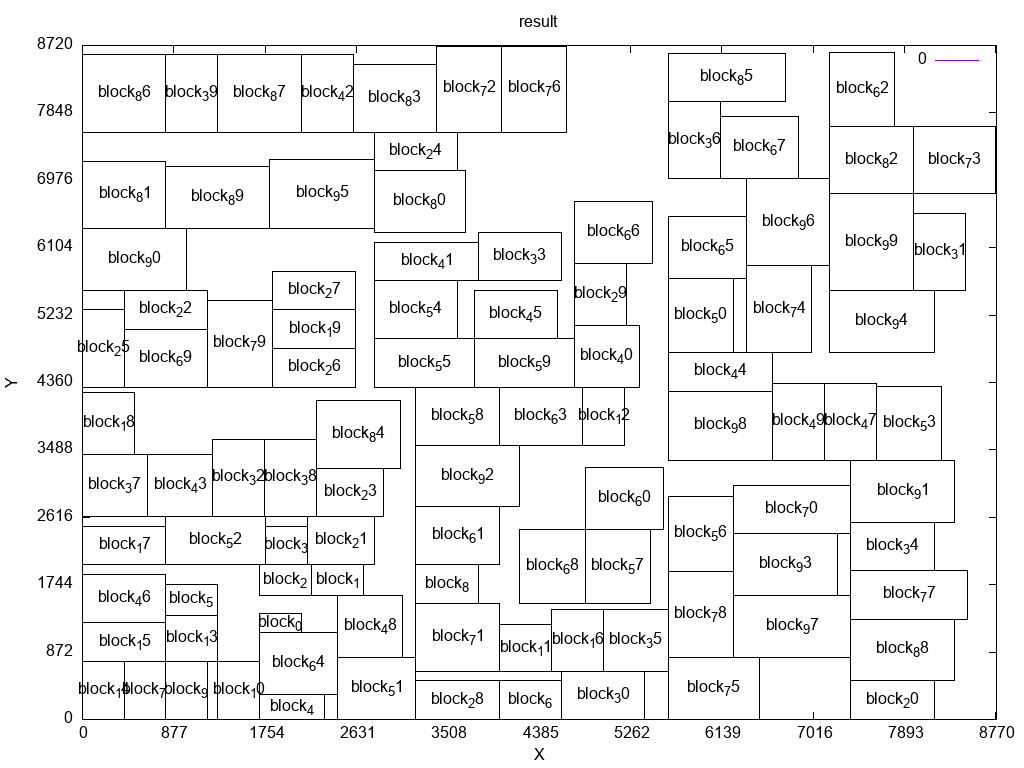
**圖(2)、左右分別是在floorplan 1000以整排與合併演算法初始化的結果。**



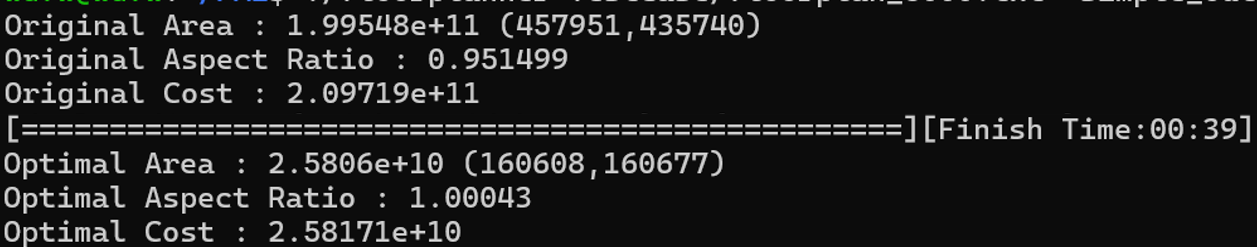
**圖(3)、左右分別是在floorplan 1000以整排與合併演算法後進行SA的結果，在數量大時，合併初始化的表現優異(因為SA移動的次數沒遠超blocks數量，使沒移動的blocks保持整串)。**

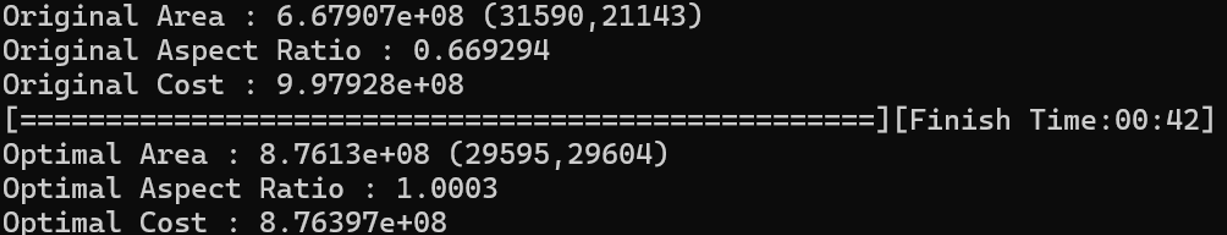


**圖(4)、左右分別是在floorplan 100以整排與合併演算法初始化的結果。**



**圖(5)、左右分別是在floorplan 100以整排與合併演算法後進行SA的結果，在數量不大下，其結果差不多(因為SA移動的次數遠超數量，所有blocks都高機率被移動幾次)。**





**圖(6)、上下分別是在floorplan 5000使用整排與合併演算法後進行SA的表現，上下的Optimal Cost差了29倍。**