PD PA2 Report

1.Data structure used in my program

因為程式內包含許多資料結構，所以我會描述程式內所運行的演算法，並說明各部份所使用的資料結構為何。

首先，我的演算法主要使用上課所教的Bstartree搭配SA尋找global optimum，且是建立於作業檔案內所附的範例往下修改與實作。此外我還寫了floorplanner.cpp與.h作為主要處理floorplan的class與對應function實作。

在construct class floorplanner時，我會先使用read_block、read_net兩個function init block、net、terminal的資訊。Block的部分，我去掉繼承terminal的屬性，將兩者設為獨立class，class terminal則照作業範例使用，我在 class floorplanner中用對應的vector將兩者記錄下來，並額外使用兩unordered_map分別作為兩者的name對data的mapping。因此我將Net中的termlist從vector<*terminal>改為vector<string>紀錄在net中的terminal的name，然後再用前述的unordered_map尋找對應terminal。

我的F-M heurstic主要都在floorplanner的function floorplan內，其中可分為以下幾步。

a. time
首先我會紀錄開始的時間，之後根據時間限制調整參數或執行方式，最後輸出結果時輸出總執行時間

b.init Bstartree
我會用隨機的方式將class Node插入樹中，Node是在樹中使用的data structure，紀錄對應的block以及在樹中的資訊，如parent、left child、right child，且為了方便控管空間問題，此處的child我用c++內建的unique_ptr處理。而class Bstartree則存root的unique_ptr及一個list，list儲存樹中的Node，主要作為方便管理Node的額外資結。

c. init solution
init完tree後，我會對tree做packing(指算出對應floor plan解)並算出cost，將tree與cost紀錄為old_tree與old_cost，若符合outline條件，同時紀錄在best_tree與best_cost中。

d.SA
初始化SA的各參數
以下是while(T > Tmin)會做的事
初始化MT, uphill = 0
以下是while(MT < 2 * N && uphill < N)會做的事 // N = k * n, n為#block, k為任意常數
(1)對tree隨機執行一move(rotate, delete&insert, swap)
(2)packing tree，計算cost與cost difference(cost - old_cost)。
(3)如果符合outline，考慮是否為best_cost，若是則紀錄為best_tree, best_cost。
(4)如果cost difference <= 0或exp(-dCost / T) > prob，則接受新解，紀錄為old_tree, old_cost，同時若cost difference > 0則uphill + 1。如果不接受新解，則將tree與cost還原成old_tree與old_cost。

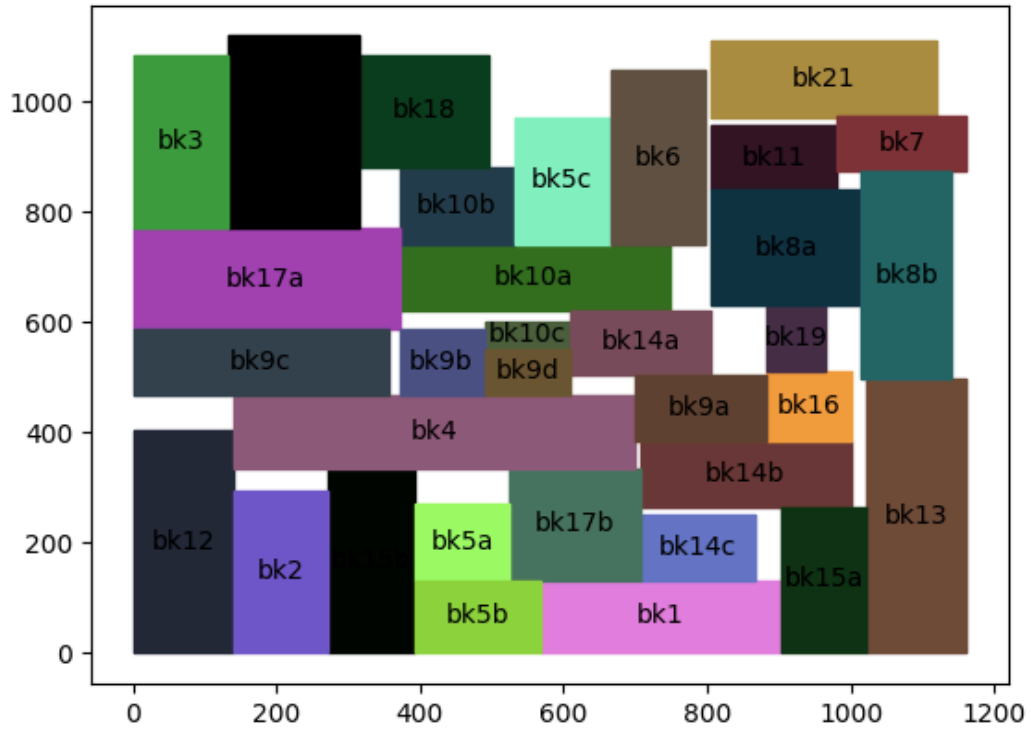
(5)MT++
end of while(MT < 2 * N && uphill < N)
T = T * R
end of while(T > Tmin)

e. calculate result
對best_tree做packing, 計算並輸出對應area, HPWL, chip width, chip height, time, 以及各block的座標。

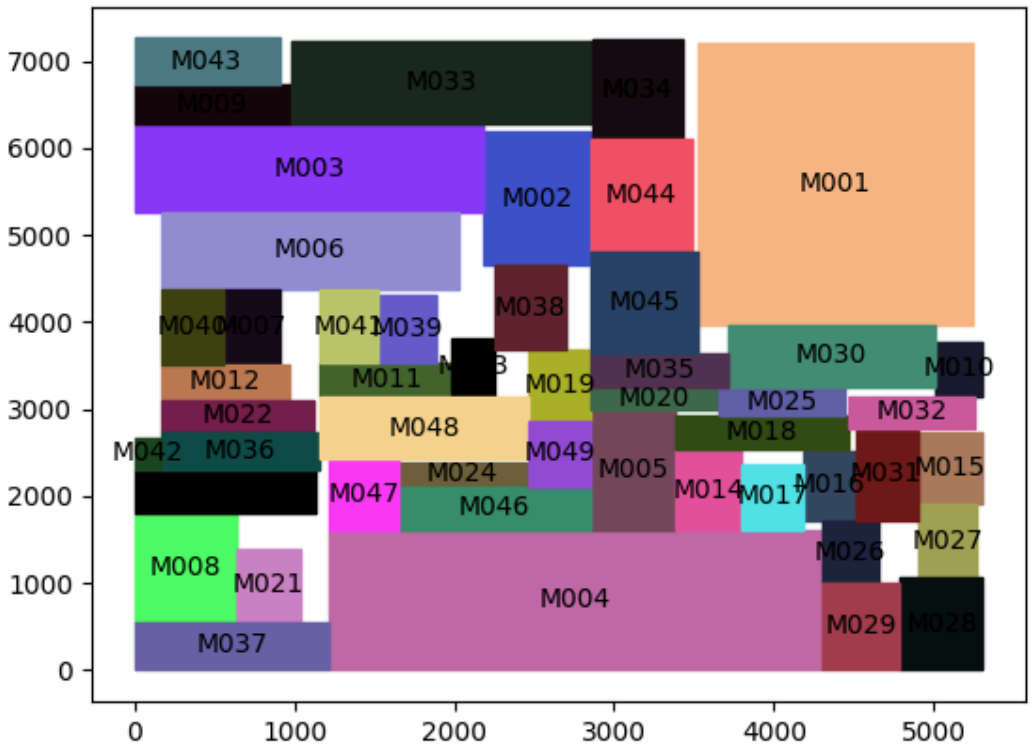
f.packing
上述為整體演算法, 此處補齊packing的細節。
packing時我會用一ContourNode做linked list, 單純紀錄horizontal contour資訊, 之後用function placeblock對root及其child recursive做對應的place。在完成整棵樹的place後, 將ContourNode delete, 放出記憶體。
placeblock做的事為, 將選定node(block)放在選定的x, 其會先根據horizontal contour算出node應放置的(x, y), 之後更新horizontal contour, 刪除多餘的node, 並對非null的child recursive call place block。

2.My findings
首先是記憶體控管問題, 這次作業的記憶體控管比較困難, 而且有很多地方要做需不需要ptr的取捨, 用了unique_ptr後有好很多, 但是unique_ptr的使用上也有很多地方要注意, 尤其是擁有權的給予與設定, 在copy Bstartree遇到不少問題。此外, 這次作業的各個隨機性都大幅度的影響最後結果, 包含找不找得到符合outline的解, 以及解的cost好壞, 在SA參數不變的情況下, 只調整Bstartree的init或move的random選擇, 就可以造成quality_score有20-30%的變化, 因此花了不少時間在調整SA參數與seed的隨機選擇。此外, case ami33的score不斷怎麼嘗試都無法找到quality_score超過4的解, 至今還沒想到解決方法, 與其他人討論也無果, 或許有時間可以再嘗試其他方法, 如fast-SA等advanced technique。最後附上各測資的floorplan圖。

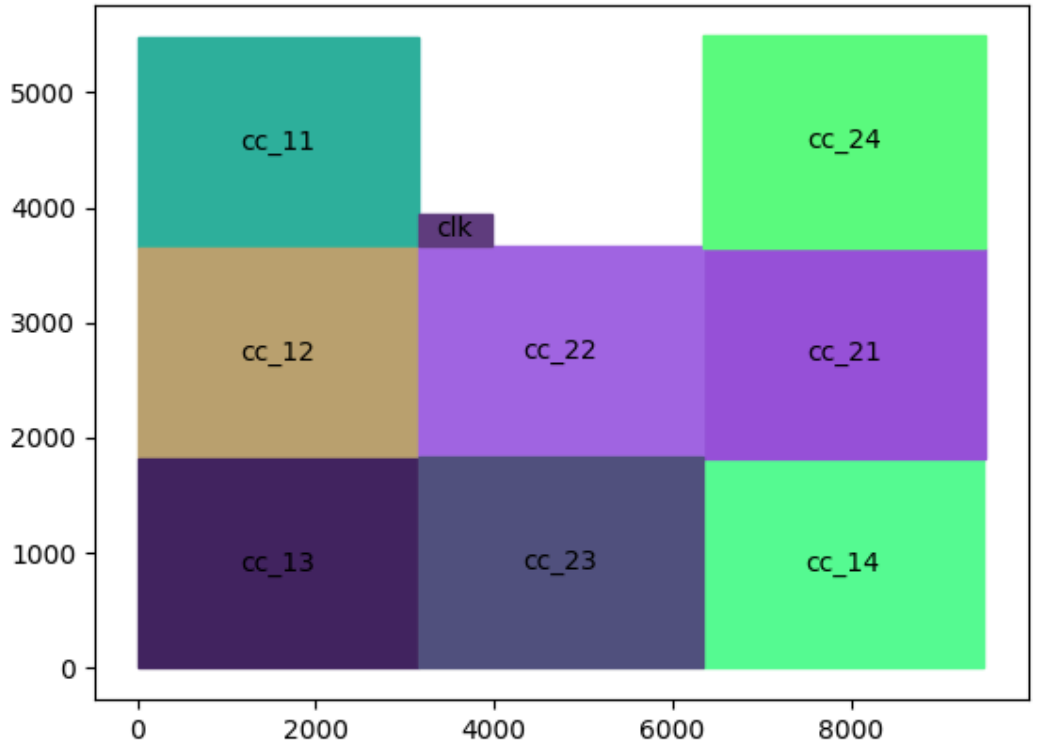
ami33:


bk3
bk18
bk21
bk6
bk5c
bk11
bk7
bk10b
bk8a
bk8b
bk17a
bk10a
bk10c
bk14a
bk19
bk9c
bk9b
bk9d
bk9a
bk16
bk4
bk14b
bk13
bk12
bk2
bk5a
bk17b
bk14c
bk15a
bk5b
bk1
0
200
400
600
800
1000
1200

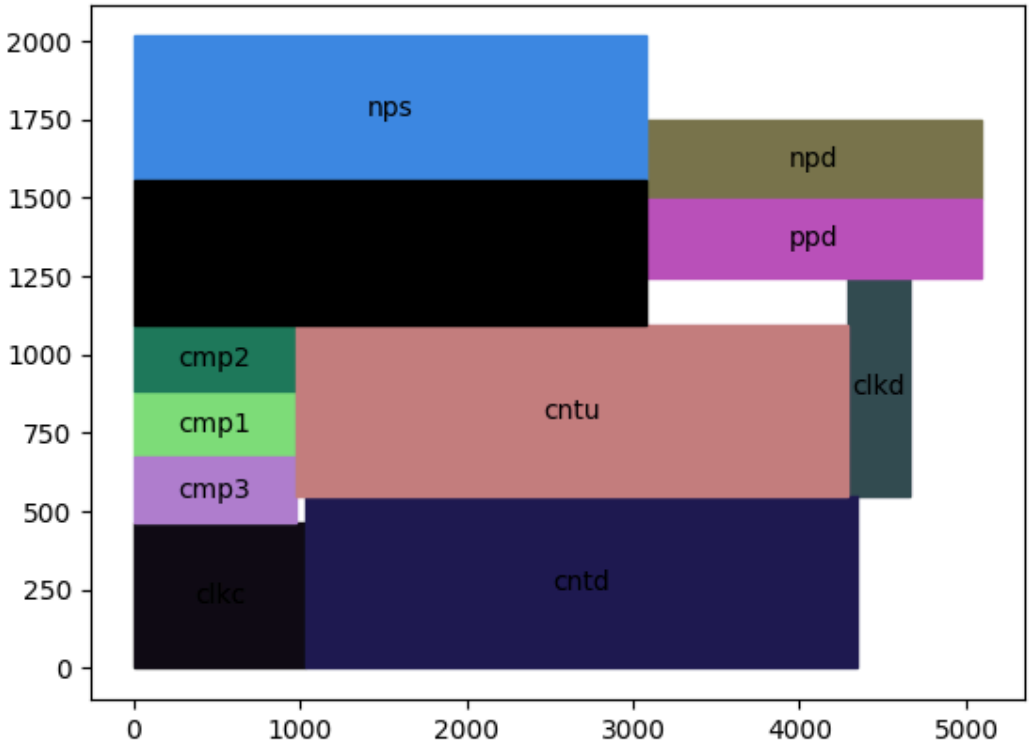
ami49:


M043
M033
M034
M009
M003
M002
M044
M001
M006
M038
M045
M040
M041
M039
M030
M010
M012
M011
M019
M035
M020
M025
M032
M022
M048
M018
M042
M036
M049
M005
M024
M031
M015
M047
M014
M017
M016
M046
M026
M027
M008
M021
M004
M029
M028
M037
0
1000
2000
3000
4000
5000
6000
7000

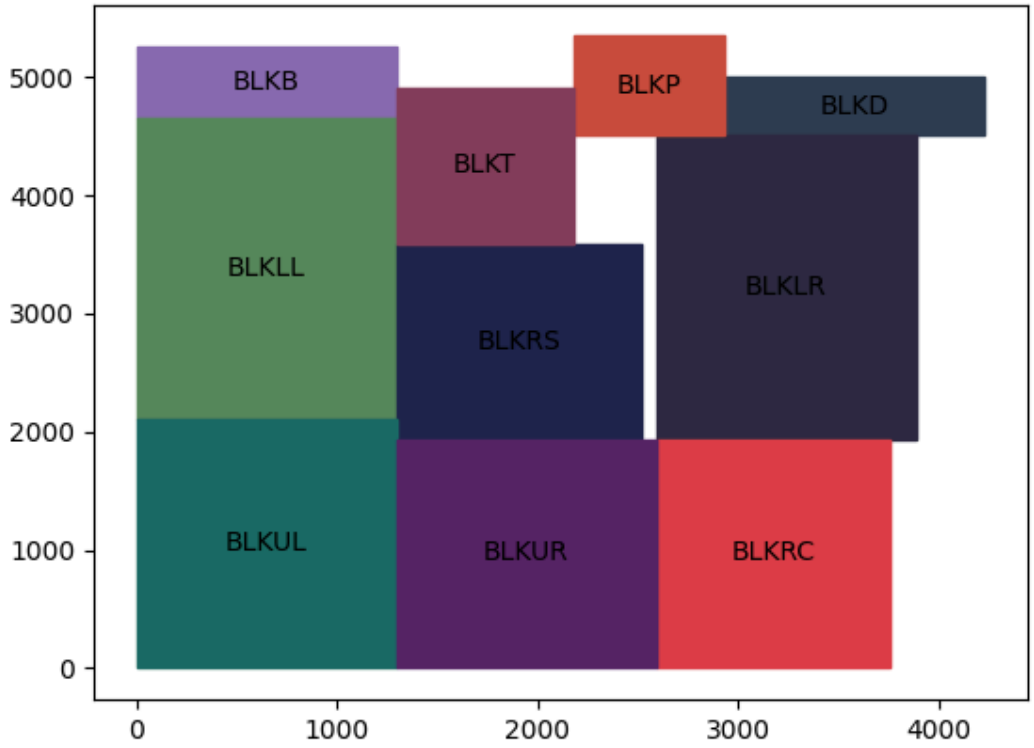
apte:


cc_11
clk
cc_24
cc_12
cc_22
cc_21
cc_13
cc_23
cc_14
5000
4000
3000
2000
1000
0
0
2000
4000
6000
8000

hp:


nps
npd
ppd
cmp2
cntu
clkd
cmp1
cmp3
clkc
cntd
2000
1750
1500
1250
1000
750
500
250
0
0
1000
2000
3000
4000
5000

xerox:


BLKB
BLKP
BLKD
BLKT
BLKLL
BLKLR
BLKRS
BLKUL
BLKUR
BLKRC
5000
4000
3000
2000
1000
0
0
1000
2000
3000
4000