CS6135 VLSI Physical Design Automation
Homework 2: Two-way Min-cut Partitioning

Bo-Wei Lee (李柏葳)

111065525

1 使用步驟與範例

1.1 編譯程式

1. 先進入 /HW2/src 的路徑。
2. 輸入 make 的指令，在 /HW2/bin 的路徑底下會產生出一個 hw2 的執行檔。
3. 輸入 make clean 的指令可以將產生的連結檔跟 bin 檔刪除。

1.2 執行程式

1. 進入 /HW2/bin 的路徑。
2. 輸入./hw2 <input file> <output file> 的指令，在指定的 output 位置檔案會產生出.out 檔。

1.3 範例

1. $ cd /HW2/src
2. $ make
3. $ cd ../bin
4. $ ./hw2 ../testcase/public1.txt ../output/public1.out

2 最終成果

	Final cut size	Read time (s)	FM process time (s)	Write time (s)	Total runtime (s)
sample	1	0.006	0	0.001	0.007
public1	114	0.008	0.005	0.001	0.014
public2	2740	0.066	0.75	0.006	0.822
public3	20233	0.43	2.352	0.02	2.802
public4	1820	0.024	0.147	0.003	0.174
public5	4876	0.217	4.061	0.011	4.289
public6	26003	1.648	28.083	0.055	29.787

Table 1: 每筆測資最終的 cut size 及運行時間

上表中每一筆結果都有通過作業裡 verifier 的驗證。運行時間皆是由 C++ chrono 函式庫裡的 high_resolution_clock 所計。

3 程式實作與優化

3.1 演算法概述

在這份作業中，我主要是用課堂所學的 Fiduccia-Mattheyses 算法進行實作。FM 演算法旨在減少不同分區之間的連接數，其特點可以在超圖模型上使用，並且有效的處理分割不平均的問題，所以很適合用在這次作業的 two-way partition 中。以圖一爲例，演算法步驟如下。

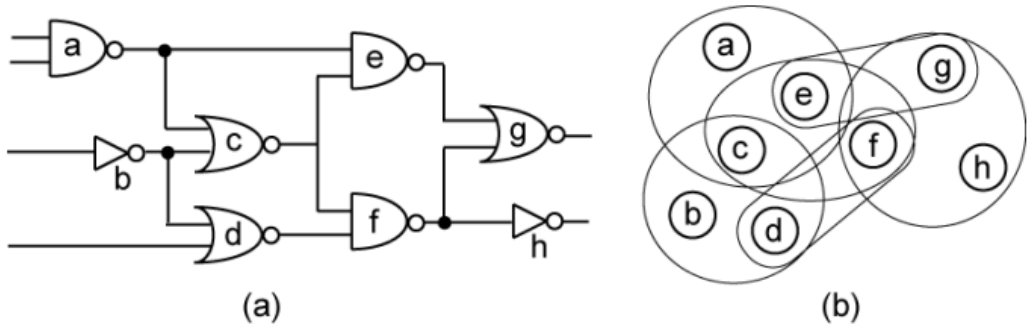

Figure 1: 範例電路

3.2 資料讀取

首先，程式會將輸入的測資作處理，並儲存成 Cell 跟 Net 兩種 class 存放到 vector 裡。以圖二爲範例，其中左邊的是超圖表示，右邊是每個 Net 存入碰到的 cell，其中 n1 包含了三個節點，a, c, e，如此類推。而 Cell 裡也會儲存其碰到的 Net。

3.3 初始化分隔圖

在這個階段，vector 裡的所有元件 (cell) 會被分成兩個 group。分配的方式是先設一個 random seed，將 vector 裡的元件打亂，然後在從 vector 的頭開始分配，哪個 group 的元件 size 目前總和比較小就把那個元件放在那邊。

3.4 計算 Gain 值

先定義兩個專有名詞: FS (i)，每當有超邊包含了元件 i，而且有且僅有 i 在元件 i 所屬的分區 (例如元組 c 在左邊)，FS (i) +1，FS (i) 初始值爲 0。TE (i)，當有超邊中所有元組都在元件 i 所屬的分區 (例如元組 c 在左邊)，TE (i) + 1，TE (0) 初始值爲 0。

舉元件 c 爲例，c 包含在超邊 n1 = a, c, e，n3 = c, f, e，n2 = b, c, d 中，其中在超邊 n3 中，只有元件 c 在左邊的集合中，其他的元件都在右邊的集合中，因此 FS (c) + 1。並沒有超邊中所有的元組都在左邊，因此 TE (c) = 0。FS (i)，每當有超邊包含了元件 i，而且有且僅有 i 在元件 i 所屬的分區 (例如元組 c 在左邊)，FS (i) + 1，FS (i) 初始值爲 0。Gain 值則爲 FS(i) + TE(i)。

3.5 建立 bucket list

在分配完 Cell 後，對於每個 Group 我都會建立一個 bucket list ，其爲一個 unordered_map，範圍爲 Pmax 至 -Pmax，其對應到的個別是一個 Cell pointer，實際上是一個 double linked list 。每個元件會依據其 Gain 值被分配到各自的位置。

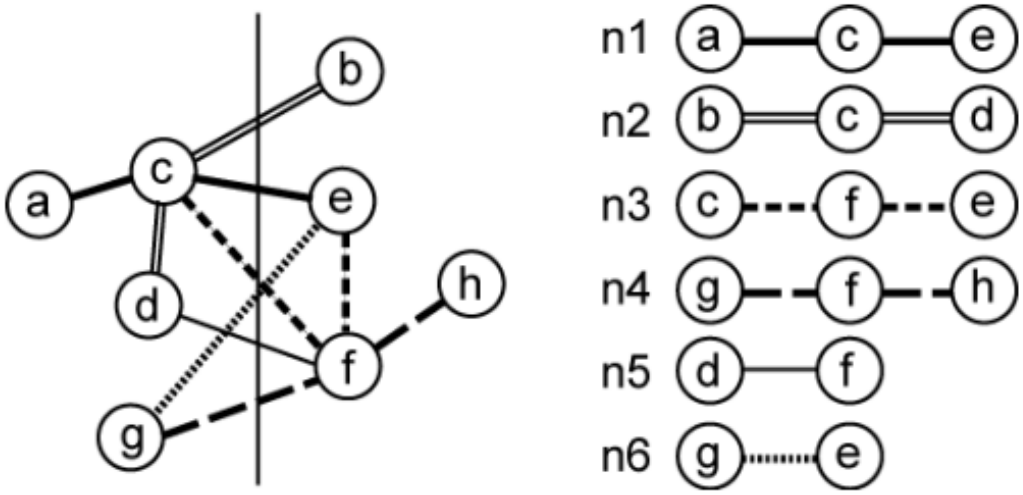

Figure 2: 在初始化完後的 partition 與 bucket list

3.6 移動 Cell

由上面第二步驟計算所有元件的 gain 值後，程式會開始選取元件進行交換。在課程中，演算法會選取具有最大 gain 的元件移動至另一個 group。在圖四中，元件 g 和元件 e 的值都是 2 是最大的，並且在移動完這個元件後，Die size 還是能遵守規範。在這裡先移動元件 e。將 e 從右邊移到左邊之後，將重新計算與 e 相關的元件的 gain 值。與 e 相關的元件集合 a, c, g, f，重新計算 Gain (a) = FS (a) - TE (a) = 0 - 1 = -1，以此類推 gain (c) = -1，gain (g) = 1 - 1 = 0，gain (f) = 2 - 0 = 2。在我這次的實作中，更新 gain 的方式一樣，但在選取元件的方式有稍微做調整，是以先一直選取 group 0 裡最大 gain 的元件移到 group 1 裡面，直到再放入元件時 group 1 超過最大面積限制，就換至將 group 1 的元件放入 group 0 中，以此類推。

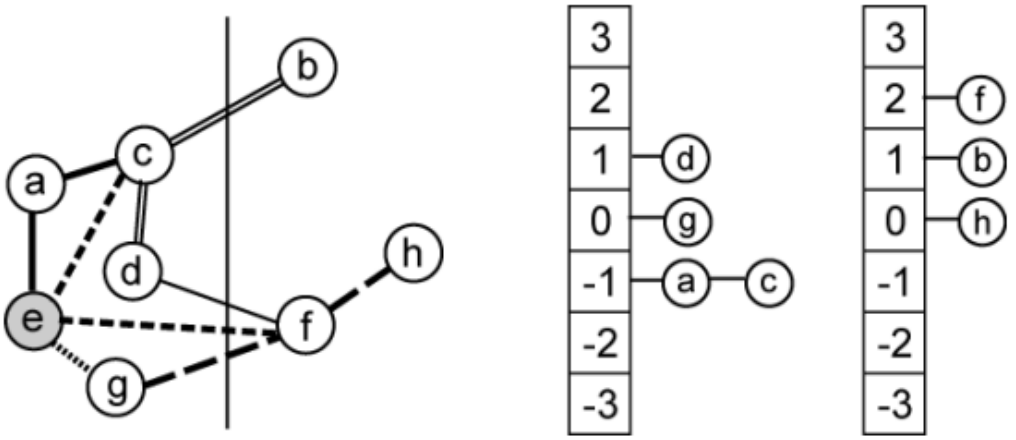

Figure 3: 做完第一次移動的模型

3.7 Partial sum 與結束條件

在每次移動元件後，partial sum 都會被更新，並且擁有最大的 partial sum 及其移動步數會被記錄下來。在 bucket list 清空後，元件會眞正的依照其記錄步數去做移動，並更新 die 的面積。在最大的 partial sum 小於零時，FM 演算法宣告結束，其結果將會被寫入.out 檔。

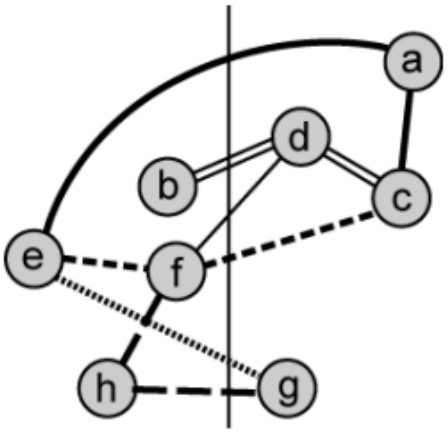

Figure 4: 在 bucket list 清空後的 graph

i	cell	$g(i)$	$\sum g(i)$	cutsize
0	-	-	-	6
1	e	2	2	4
2	***d***	**1**	**3**	**3**
3	***b***	**0**	**3**	**3**
4	***g***	**0**	**3**	**3**
5	a	-1	2	4
6	f	-1	1	5
7	h	0	1	5
8	c	-1	0	6

Figure 5: 在移動所有元件後的 graph，可以看到在移動第二步後到第四步後 cutsize 最小

3.8 程式優化

在這份程式，絕大部分的地方都是按照剛剛所提的 FM 演算法去做實做的，比較不同的地方主要有兩個。第一個是在選取元件移動的策略有做過調整，詳情在 3.6 。經過實驗發現在速度以及 cutsize 上有比選取目前最大 gain 的元件還要好一點。第二個是在一開始初始化 partition 的方式，我總共有試過幾種不同的 partition 方式，包含先將 group 0 填滿再填 group 1，把 pin 較少的 cell 放在同一邊等方式。在最後是發現用 3.2 的效果最好。推測原因是因爲那些大的元件較平均分散在兩邊時，比較容易不會因爲元件超過 Die size 的限制而無法以最佳解交換。另外我也有試好幾種 random seed 的數值，看哪一個 random seed 會讓 partition 完的平均 cutsize 最小。

3.9 程式平行化

我沒有實做這部分，因爲每一個 bucket list 的 iteration 都需要上一步的結果，這部分無法平行化。在移動 bucket list 的元件時也許可以找出好幾個沒有互相有關聯的元件去在不同的 thread 做移動與更新 gain，但本人認爲找出不會互相影響的元件的 overhead，可能會比節省的時間還多，也划不來。

4 討論與心得

這次作業最核心也最讓人頭痛的地方莫過於制定資料結構了。雖然在講義裡就有詳細的介紹演算法裡的 bucket list，在實作起來就又是另一回事了。一開始我其實是選用 2d vector 來儲存元件的，但後來發現在移動元件時會遇到一些問題，例如在移除元件時需要另外標記這個位置的 node 無法使用，這些瑣碎的小細節加起來反而影響程式的整潔性，後來才使用 map 對應至 cell 的 double linked list。另一方面，在制定哪些 class 需要哪些 attribute 時也是需要使用到時才慢慢補上去的。在開發時，由於程式碼需要的功能太複雜，本來習慣 one-filer 的我也嘗試使用了一些 OOP 的概念到程式中，包含使用 class 來包裹變數與函式、撰寫 header 來制定 API 等、將不同功能的 class 放到不同資料夾等。