利用 MapRedcue 架構列舉所有最小生成樹

蔡宏偉[†]王弘倫[‡]
國立臺北商業技術學院
資訊與決策科學研究所

摘要

在一個邊有權重的無向連通圖裡，最小生成樹有可能不是唯一的，一個無向連通圖中可能找到一個或多個不同的最小生成樹，隨著無向連通圖裡的節點與邊增多，所有最小生成樹的數量也有可能會增多，在本文章中利用 *MapRedcue* 架構，使用多台電腦平行處理一個無向連通圖來找出所有的最小生成樹，透過實驗，探討使用 *MapReduce* 架構來平行找出所有的最小生成樹之效率。

關鍵詞: 最小生成樹 (minimum spanning tree), 列舉演算法 (enumerating algorithm), Hadoop, MapRedcue

1 緒論

列舉最小生成樹的課題從以前就一直有人在討論，在 1995 年 Kapoor 和 Ramesh 提出了一個列舉生成樹的方法 [5]，同年，Eppstein 提出了一個令邊有權重的無向連通圖轉化成一個只包含最小生成樹的無向連通圖 (equivalent graph) [3]，利用 Kapoor 和 Ramesh 提出的列舉生成樹的方法，結合出一個解決列舉最小生成樹的方法，時間複雜度在 $O(m + n\log n + k)$，k 表示最小生成樹數量，m 表示邊的數量，n 表示圖的節點數量。在 2010 年，Yamada、Kataoka 和 Watanabe 也提出了一個列舉最小生成樹的方法 [8]，時間複雜度為 $O(km\log n)$。

不論是在網路設計 [7]，或是分子流行病學 (molecular epidemiology) [6] 的研究等，都會要尋找 MST，但是在一個無向連通圖裡面，MST 可能不是唯一的，而不同的最小生成樹可能具有不同的特性。令一個無向連通圖 $G = (V, E)$，頂點的集合 $V = \{v_1, ..., v_n\}$ 和邊的集合 $E = \{e_1, ..., e_m\}$，每個邊 e 的權重 $w(e) > 0$，對任意生成樹 $T = \{e_{i_1}, \ldots, e_{i_{n-1}}\}$，定義 $w(T) = \sum_{j=1}^{n-1} w(e_{i_j})$，若 $T = \emptyset$，則 $w(T) = \infty$。在 n 個點且邊權重都相同的完全圖 (complete graph) 中，由 Cayley 公式 [1] 可知該圖的 MST 個數為 n^{n-2}。

本論文結構如下，在第 2 節裡，我們簡單介紹 MapReduce 的歷史與架構，在第 3.1 節裡，介紹了一個列舉所有最小生成樹的演算法，在第 3.2 節裡，則是利用 MapReduce 實現了 3.1 節的演算法，在第 4 節，我們呈現了實驗的結果與相關的探討。

2 MapRedcue 介紹

MapReduce 起源於 2004 年發表的文章 [2]，描述在大規模叢集上簡化資料處理，MapReduce 是一個藉由平行運算架構同時運算大量資料，利用 Map 任務與 Reduce 任務來完成運算。Map 任務是把一個任務拆成多個子任務進行運算，Reduce 任務是指將運算結果重新組合、建立索引後再送回節點。MapRedcue 的叢集運算是先將一台電腦設定為 Master 負責監控，其餘電腦設為 Worker node 負責運算，將資料分割成固定大小在丟給 MapReduce 作業，稱為輸入分割 (inputSpilts)，每個分割裡又有多行紀錄 (record)，並且會針對每個分割建立一個 Map 任務，分割中的每一筆記錄 (record)，會交給自訂的 map 函數來處理。map 函數接收一個 $< key1, value1 >$ 形式的輸入 (圖 1)，然後同樣產生一個 $list < key2, value2 >$ 的中間輸出，然後 MapReduce 會將所有具有相同中間 key 值的 $value1$ 集合一起傳遞給 $reduce$ 函數，$reduce$ 函數接收一個 $< key2, list(value2) >$ 形式的輸入，最後輸出 $< key3, value3 >$。

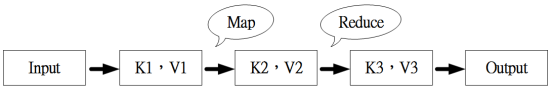

圖 1: MapReduce 程式資料變化過程

[†]s10176109@student.ntcb.edu.tw
[‡]corresponding author:hlwang@webmail.ntcb.edu.tw

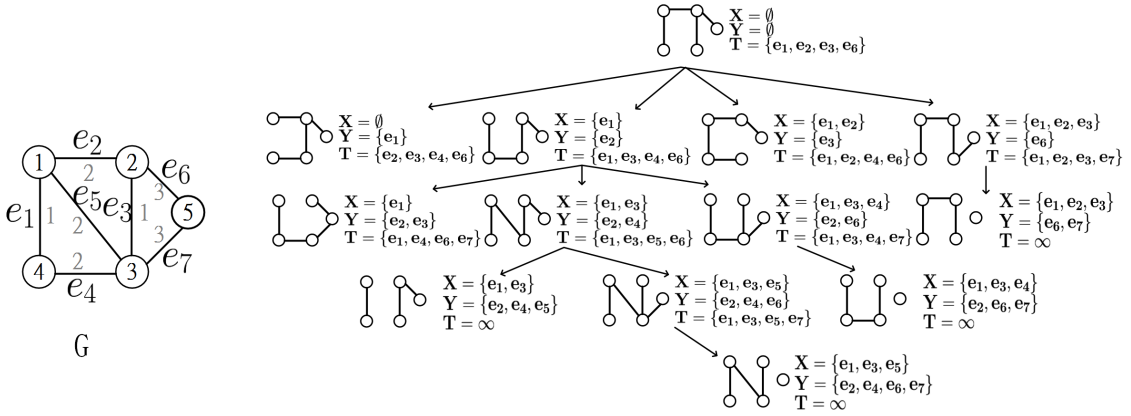

圖 2: 連通圖 G ，邊旁的數字表示為該邊權重與 ListMST(G) 執行過程對應遞迴樹

3 列舉最小生成樹的方法與利用 MapReduce

3.1 最小生成樹的列舉演算法

本論文中列舉最小生成樹的方法，其概念與某些文章提出的方法類似 [3,4,5,8]，為一遞迴演算法，我們定義為 ListMST(G)，且需要兩個方法，Kurskal(G, X, Y) 與 ListMST(G, X, Y, T^*)，詳細描述如下：

- Kurskal(G, X, Y)：用來計算圖 G 的一個 MST，且滿足 $X \subseteq T, Y \cap T = \emptyset$，如果 T 不存在，Kurskal(G, X, Y) 就回傳 $\emptyset$，Kurskal(G, X, Y) 方法可以利用傳統的 Kurskal 演算法 [4] 加以改寫。
- ListMST(G, X, Y, T^*)：列舉圖 G 中所有最小生成樹 T，且滿足 $X \subseteq T, Y \cap T = \emptyset$ 和 $w(T) \leq w(T^*)$，如果 T 不存在，就停止。對任意 $T = \{e^1, \ldots, e^{n-1}\}$，我們不失一般性假設，$X = \{e^1, \ldots, e^k\}$，此虛擬碼如 **Algorithm 1**。

Algorithm 1 ListMST(G, X, Y, T^*)：

```
1: {e^1, ..., e^{n-1}} ← Kurskal(G, X, Y)
2: T ← {e^1, ..., e^{n-1}}
3: if (w(T) > w(T*) or w(T) = ∞)
4:     return
5: else
6:     for (i ← k + 1 to n − 1)
7:         ListMST(G, X ∪ {e^1, ..., e^{i−1}}, Y ∪ {e^i}, T*)
8:     return T
```

在定義 ListMST(G, X, Y, T^*) 和 Kurskal(G, X, Y) 後，我們即可利用其遞迴定義 ListMST(G) 如 **Algorithm 2**。

Algorithm 2 ListMST(G)：

```
1: T* ←Kurskal's(G, ∅, ∅)
2: ListMST(G, ∅, ∅, T*)
3: return T
```

ListMST(G) 執行過程如圖 2 所示。

3.2 利用 MapReduce 列舉最小生成樹

在本節中為了閱讀上的方便，我們用 IN 與 OUT 這兩個集合表示 3.1 節的 X 與 Y 集合，由 3.1 節的方法中，我們可觀察出每一次遞迴產生出的子問題都是獨立的，亦即遞迴樹中任一節點下所出產生出 MST 皆不相同，因此在利用 MapReduce 架構實現 **Algorithm 1** 時，我們僅需利用 map 函數來達到我們的目的，如圖 3 所示。

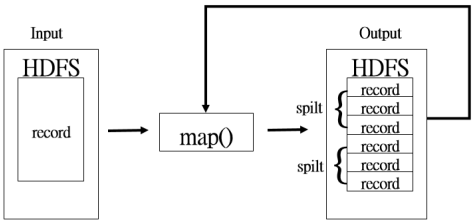

圖 3: MapReduce 執行流程

在處理過程上，其概念是以廣度優先搜尋順序迭代處理遞迴樹，在設計演算法時，我們欲將每一個遞迴樹上的節點當作一 record 處理，即在每個 map 函數中實現 **Algorithm 3**，在鍵值對 $< key, value >$ 的設定上，我們

將所需資訊都紀錄在 $value$ 中，key 的為可任意設定。針對節點 P，其對應的 map 函數輸入的 value 為 P 所對應的 G, IN, OUT, T^*，map 函數輸出的 $value$ 為 P 的子節點對應之 G, IN, OUT, T^*，每個 record 的格式，設定如下：

$$G < \text{tab} > IN < \text{tab} > OUT < \text{tab} > w(T^*)，$$

其中 T^* 為圖 G 的 MST。基於以上設定，我們可定義 map 函數如 **Algorithm 3**。

Algorithm 3 MapTask< $key, value$ >:

輸入： 鍵值組對 $< key, value >$，key 為任意值，$value$ 是 input 檔案上的一行資料 (record)，此行資料包含 $G, IN, OUT, w(T^*)$。

輸出： 鍵值組對 $< key, value >$，key 為 $NULL$，$value$ 是包含多行 record 的檔案，此檔案每行 record 包含 $G, IN, OUT, w(T^*)$。

1: $\{e^1, \ldots, e^{n-1}\} \leftarrow$，Kurskal$(G, X, Y)$
2: $T \leftarrow \{e^1, \ldots, e^{n-1}\}$
3: **if** $(w(T) > w(T^*)$ or $w(T) = \infty)$
4: **return**
5: **else**
6: **for**$(i \leftarrow k+1$ **to** $n-1)$
7: $key \leftarrow \emptyset$
8: $value \leftarrow (G, X \cup \{e^1, \ldots, e^{n-1}\}, Y \cup \{e^i\}, T^*)$
9: write$(key, value)$
10: output T //將 $value$ 寫為 record
11: **return**

初始執行時，IN 與 OUT 皆設為空集合，T^* 為圖 G 的 MST，為迭代地執行一個 map 函數，在執行過程中 map 函數可能會持續產生一些新的 record，而每個 record 會在下回合持續進行處理，直到所有 record 處理完為止，如圖 4 所示。

4 實驗結果

本次實驗，將針對第 3 節所設計的演算法來進行叢集式 Hadoop 實驗，使用 MapRedcue 與 HDFS (分散式檔案系統)，Hadoop 版本為 1.0.3，我們在研究教室裡部屬 11 台電腦來運算，1 台為 Master，10 台為 DataNode，每台電腦的電腦規格皆相同，3.40GHz 四核心 Intel Core i7、記憶體 4GB、磁碟容量 50.4G、作業系統為 Windows 7，10 台 DataNode 皆連到 Juniper EX2200 1Gps 全雙工乙太網路交換器。

本次測試採用所有邊權重皆為 1 的完全圖且完全圖節點數為 7、8、9、10。圖的紀錄方式如下:

weight<tab>node1<tab>node2，

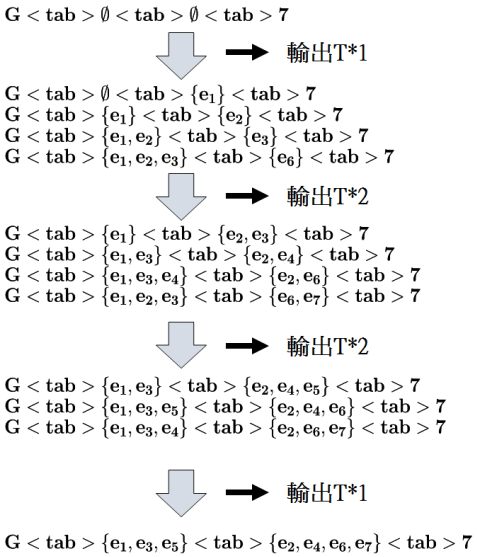

圖 4: 利用 MapReduce 列舉 MST 之流程。G 為圖 2 之圖例。

其中 node1 與 node2 為圖上某邊之兩端點，weight 為該邊之權重。以圖 2 為例，其中之圖 G 紀錄如下。

1<tab>1<tab>4
1<tab>2<tab>3
2<tab>1<tab>2
2<tab>1<tab>3
2<tab>3<tab>4
3<tab>2<tab>5
3<tab>3<tab>5

檔案切割方式 (InputFormat) 設定為 NLineInputFormat(其中 N=1000，即一千個 record 為一個 Map 任務)，所有實驗皆已 1、5、10 台叢集式電腦進行，從圖 5 可以看出當完全圖節點數為 7 時，連結的電腦數量並沒有明顯得在處理速度上提升，這是因為 MST 的產生數量並未很多，也代表 record 數量並不是很多，當連結 10 台電腦時並未完全使用到全部電腦做運算，當節點數為 9 時即有明顯看出處理速度有大幅提升，到節點為 10 時，10 台電腦處理速度即接近到 5 台電腦處理速度的兩倍。由圖 6 可看出，在計算 MST 某範圍內，計算時間的增長率有趨於平緩的現象，但由圖 7 可看出，每秒所產生的 MST 數量差距會越來越小，由此我們可預期 MST 數量達到一定範圍後，每秒所產生的 MST 數量會趨於定值，藉此我們可以去預估 11 個點的完全圖，其 MST 數量的計算時間。在 11 個點的完全圖中，其 MST 的數量 10 個點的完全圖之 MST 數量多出約 23.5

倍，由圖 7 所提供的資訊，可推斷出 Hadoop 叢集之電腦數量為十台時約要花費 33 天。

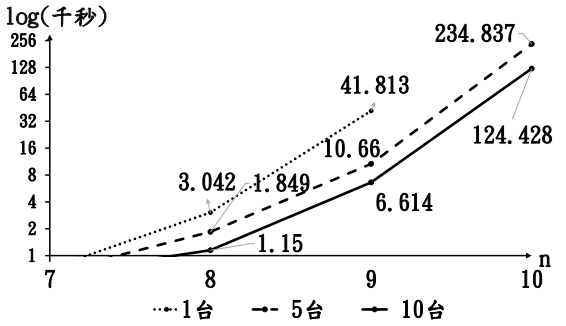

圖 5: 各叢集規模下 $n = 7, 8, 9, 10$，產生完全圖之所有生成樹所耗費時間。

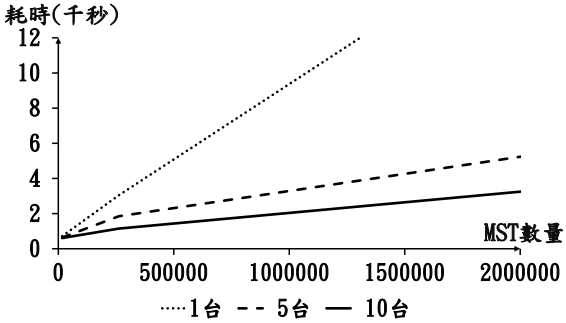

圖 6: MST 個數與相對應計算時間之關係圖。

node值	MST數量	時間(秒)	1秒產出的MST數量
7	16807	614	27.372
8	262144	1150	227.951
9	4782969	6614	723.158
10	100000000	124428	803.677

圖 7: Hadoop 電腦叢集為 10 台電腦，每秒輸出的 MST 數量。

5 結論

本研究提出一個利用 MapReuce 架構列舉所有最小生成樹的方法，並以邊權重皆相同之完全圖做為測試對象。透過此次實驗可知，數量上未超過 10^8 個 MST，約能再 36 小時內計算完成，未來除了進一步增進列舉演算法之效率外，也期望針對某些實際數據進行相關實驗。

參考文獻

[1] A. Cayley. *A theorem on trees.* Quart. J. Math. 23 (1889) 376–378.

[2] J. Dean and S. Ghemawat. *MapReduce: simplied data processing on aarge clusters.* Appeared in: OSDI'04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA, 2004.

[3] D. Eppstein. *Representing all minimum spanning trees with applications to counting and generation.* Tech. Report 95-50, Department of Information and Computer Science, University of California, Irvine, 1995.

[4] J.B. Kruskal, *On the shortest spanning subtree of a graph and the traveling salesman problem.* Proc. Am. Math. Soc. 7 (1956), 8–50.

[5] S. Kapoor and H. Ramesh. *Algorithms for enumerating all spanning trees of undirected and weighted graphs.* SIAM J. Comput. 24 (1995), 247–265.

[6] J. Salipante and G. Hall.*Inadequacies of minimum spanning trees in molecular epidemiology.* J. Clin. Microbiol. 49 (2011). 3568–3575.

[7] H. Cormen, E. Leiserson, L. Rivest and C. Stein. *Introduction to algorithms, second edition.* MIT Press, Cambridge, 2001.

[8] T. Yamada, S. Kataoka and K. Watanabe. *Listing all the minimum spanning trees in an undirected graph.* Int. J. Comput. Math. 87 (2010), 3175–3185.