



國立成功大學 資工所

Data Mining

資料探勘

**Project 3**

**Link Analysis**

課程教授：高宏宇

學生：葉芯妤

學號：P96074147



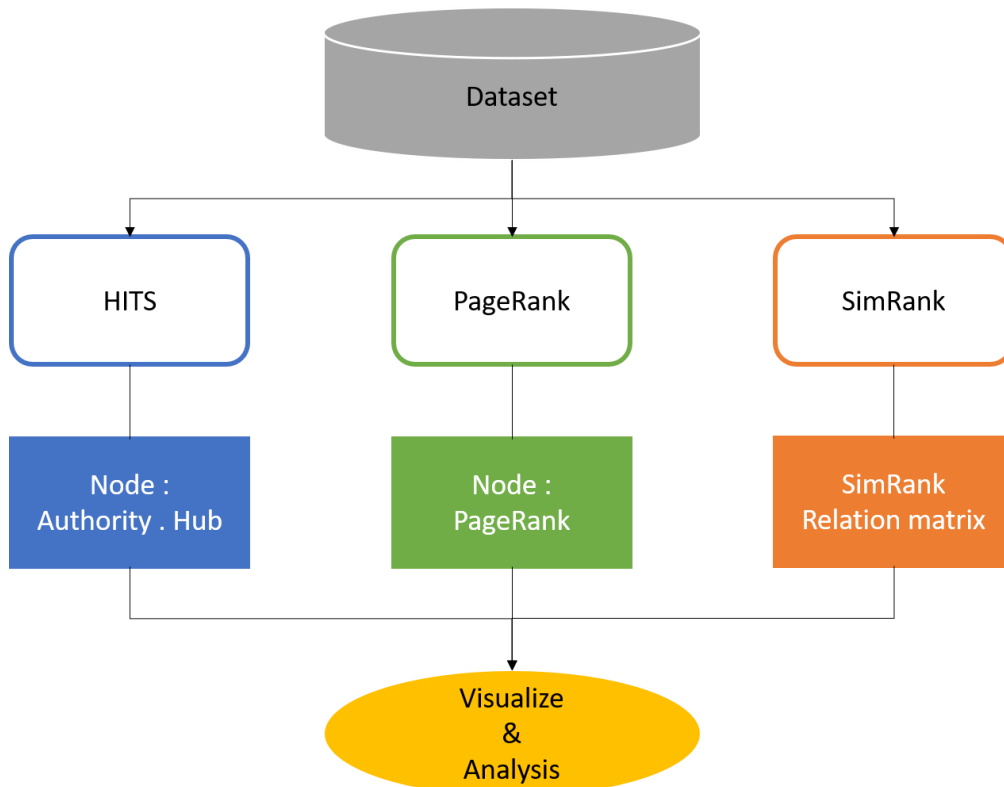
# 目錄

一、專案內容簡介 .....	3
二、程式實作 .....	3
三、結果分析與討論 .....	6
四、運算效能分析 .....	13
五、結論.....	13

## 一、 簡介

測試資料集包含 8 個 graph data，皆以文字檔形式(.txt)儲存，包含 6 個 dataset、1 個 Project1 的交易資料集，以及 1 個從 Project 1 產生的關聯法則推導的 graph。

本專案實作三個 Link Analysis 的演算法，分別為 HITS、PageRank 以及 SimRank，前兩個主要用途為分析網頁點閱排名的方法，後者為分析網頁之間的相似度，藉由上述三個演算法分析 8 個資料集，專案架構如下圖一所示。



圖一、專案架構

## 二、 程式實作

### A. 環境

作業系統：Window 10

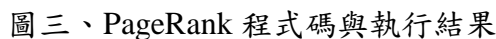
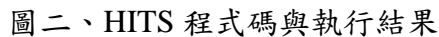
程式語言：Python3.6

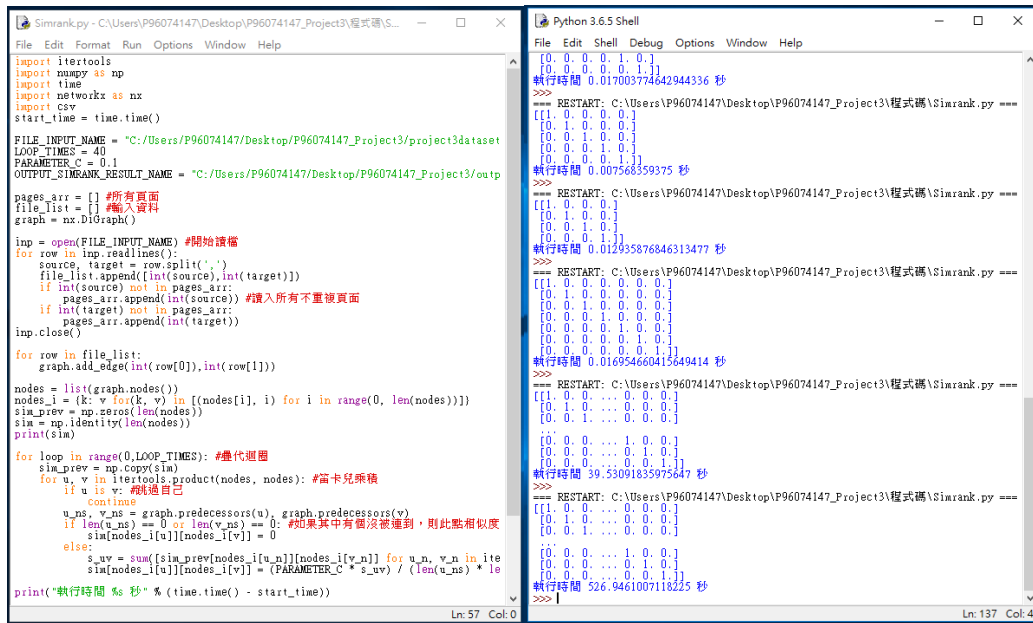
額外工具：WEKA、Excel

### B. 程式碼

本專案參考 github 以 Python 撰寫 HITS、PageRank 以及 SimRank 三個演算法的程式碼，輸入預分析的鏈結圖形資料，經過三個圖一節點 Link Analysis 演算

下圖二、三和四分別為 HITS、PageRank 以及 SimRank 的程式碼與執行結果，紅色中文字為對程式碼所作的註解。而圖五為為了方便分析，將 Project1 的交易資料集轉換為 TXT 檔形式的程式碼。





```
SimRank.py - C:\Users\P96074147\Desktop\P96074147_Project3\程式碼\...
File Edit Format Run Options Window Help

import itertools
import numpy as np
import time
import networkx as nx
import csv
start_time = time.time()

FILE_INPUT_NAME = "C:/Users/P96074147/Desktop/P96074147_Project3/project3dataset"
LOOP_TIMES = 40
PARAMETER_C = 0.1
OUTPUT_SIMRANK_RESULT_NAME = "C:/Users/P96074147/Desktop/P96074147_Project3/outp

pages_arr = [] #所有頁面
file_list = [] #輸入資料
graph = nx.DiGraph()

inp = open(FILE_INPUT_NAME) #開始讀檔
for row in inp.readlines():
    source, target = row.split(',')
    file_list.append((int(source), int(target)))
    if int(source) not in pages_arr:
        pages_arr.append(int(source)) #讀入所有不重複頁面
    if int(target) not in pages_arr:
        pages_arr.append(int(target))
inp.close()

for row in file_list:
    graph.add_edge(int(row[0]), int(row[1]))

nodes = list(graph.nodes())
nodes_i = [k: v for (k, v) in [(nodes[i], i) for i in range(0, len(nodes))]]
sim_prev = np.zeros((len(nodes)))
sim = np.identity(len(nodes))
print(sim)

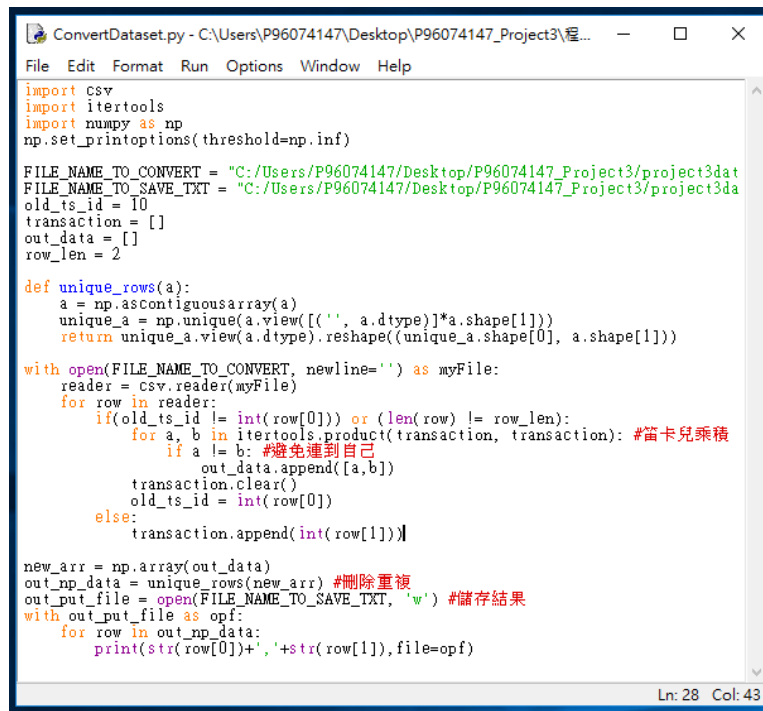
for loop in range(0, LOOP_TIMES): #疊代迴圈
    sim_prev = np.copy(sim)
    for u, v in itertools.product(nodes, nodes): #當卡兒乘積
        if u is v: #跳過自己
            continue
        u_ns, v_ns = graph.predecessors(u), graph.predecessors(v)
        if len(u_ns) == 0 or len(v_ns) == 0: #如果其中有個沒被連到，則此點相似度
            sim[nodes_i[u]][nodes_i[v]] = 0
        else:
            s_uv = sum([sim_prev[nodes_i[u_n]][nodes_i[v_n]] for u_n, v_n in ite
sim[nodes_i[u]][nodes_i[v]] = (PARAMETER_C * s_uv) / (len(u_ns) * le

print("執行時間 %s 秒" % (time.time() - start_time))
Ln: 57 Col: 0

Python 3.6.5 Shell
File Edit Shell Debug Options Window Help

[[0.0, 0.0, 0.1, 0.0]]
[[0.0, 0.0, 0.0, 1.0]]
執行時間 0.017003774442944336 秒
>>>
==== RESTART: C:\Users\P96074147\Desktop\P96074147_Project3\程式碼\SimRank.py ====
[[1.0, 0.0, 0.0, 0.0]]
[[0.1, 0.0, 0.0, 0.0]]
[[0.0, 1.0, 0.0, 0.0]]
[[0.0, 0.1, 0.0, 0.0]]
[[0.0, 0.0, 1.0, 0.0]]
[[0.0, 0.0, 0.1, 0.0]]
[[0.0, 0.0, 0.0, 1.0]]
執行時間 0.007568359375 秒
>>>
==== RESTART: C:\Users\P96074147\Desktop\P96074147_Project3\程式碼\SimRank.py ====
[[1.0, 0.0, 0.0, 0.0]]
[[0.1, 0.0, 0.0, 0.0]]
[[0.0, 1.0, 0.0, 0.0]]
[[0.0, 0.1, 0.0, 0.0]]
[[0.0, 0.0, 1.0, 0.0]]
[[0.0, 0.0, 0.1, 0.0]]
[[0.0, 0.0, 0.0, 1.0]]
執行時間 0.012935876846313477 秒
>>>
==== RESTART: C:\Users\P96074147\Desktop\P96074147_Project3\程式碼\SimRank.py ====
[[1.0, 0.0, 0.0, 0.0, 0.0]]
[[0.1, 0.0, 0.0, 0.0, 0.0]]
[[0.0, 1.0, 0.0, 0.0, 0.0]]
[[0.0, 0.1, 0.0, 0.0, 0.0]]
[[0.0, 0.0, 1.0, 0.0, 0.0]]
[[0.0, 0.0, 0.1, 0.0, 0.0]]
[[0.0, 0.0, 0.0, 1.0, 0.0]]
[[0.0, 0.0, 0.0, 0.1, 0.0]]
[[0.0, 0.0, 0.0, 0.0, 1.0]]
執行時間 0.016954660415649414 秒
>>>
==== RESTART: C:\Users\P96074147\Desktop\P96074147_Project3\程式碼\SimRank.py ====
[[1.0, 0.0, ... 0.0, 0.0, 0.0]]
[[0.1, 0.0, ... 0.0, 0.0, 0.0]]
[[0.0, 1.0, ... 0.0, 0.0, 0.0]]
[[0.0, 0.1, ... 0.0, 0.0, 0.0]]
[[0.0, 0.0, ... 1.0, 0.0, 0.0]]
[[0.0, 0.0, ... 0.1, 0.0, 0.0]]
[[0.0, 0.0, ... 0.0, 1.0, 0.0]]
[[0.0, 0.0, ... 0.0, 0.1, 0.0]]
[[0.0, 0.0, ... 0.0, 0.0, 1.0]]
執行時間 39.53091835975647 秒
>>>
==== RESTART: C:\Users\P96074147\Desktop\P96074147_Project3\程式碼\SimRank.py ====
[[1.0, 0.0, ... 0.0, 0.0, 0.0]]
[[0.1, 0.0, ... 0.0, 0.0, 0.0]]
[[0.0, 1.0, ... 0.0, 0.0, 0.0]]
[[0.0, 0.1, ... 0.0, 0.0, 0.0]]
[[0.0, 0.0, ... 1.0, 0.0, 0.0]]
[[0.0, 0.0, ... 0.1, 0.0, 0.0]]
[[0.0, 0.0, ... 0.0, 1.0, 0.0]]
[[0.0, 0.0, ... 0.0, 0.1, 0.0]]
[[0.0, 0.0, ... 0.0, 0.0, 1.0]]
執行時間 526.9461007118225 秒
>>>
```

圖四、SimRank 程式碼與執行結果



```
ConvertDataset.py - C:\Users\P96074147\Desktop\P96074147_Project3\程...
File Edit Format Run Options Window Help

import csv
import itertools
import numpy as np
np.set_printoptions(threshold=np.inf)

FILE_NAME_TO_CONVERT = "C:/Users/P96074147/Desktop/P96074147_Project3/project3dat
FILE_NAME_TO_SAVE_TXT = "C:/Users/P96074147/Desktop/P96074147_Project3/project3da
old_ts_id = 10
transaction = []
out_data = []
row_len = 2

def unique_rows(a):
    a = np.ascontiguousarray(a)
    unique_a = np.unique(a.view(['', a.dtype])*a.shape[1])
    return unique_a.view(a.dtype).reshape((unique_a.shape[0], a.shape[1]))

with open(FILE_NAME_TO_CONVERT, newline='') as myFile:
    reader = csv.reader(myFile)
    for row in reader:
        if (old_ts_id != int(row[0])) or (len(row) != row_len):
            for a, b in itertools.product(transaction, transaction): #當卡兒乘積
                if a != b: #避免連到自己
                    out_data.append([a, b])
            transaction.clear()
            old_ts_id = int(row[0])
        else:
            transaction.append(int(row[1]))

new_arr = np.array(out_data)
out_np_data = unique_rows(new_arr) #刪除重複
out_put_file = open(FILE_NAME_TO_SAVE_TXT, 'w') #儲存結果
with out_put_file as ofp:
    for row in out_np_data:
        print(str(row[0])+' '+str(row[1]), file=ofp)
Ln: 28 Col: 43
```

圖五、CovertDataset 程式碼

三個演算法程式當中，除了 SimRank 之外，進行演算法迭代時皆使用設定容錯常數(ERROR\_TOLERANCE)判定迴圈何時結束，SimRank 資料較大量時分析速度會大幅降低，由於輸入資料皆為小量，所以單純以設定其迴圈的迭代次數來判斷收斂。

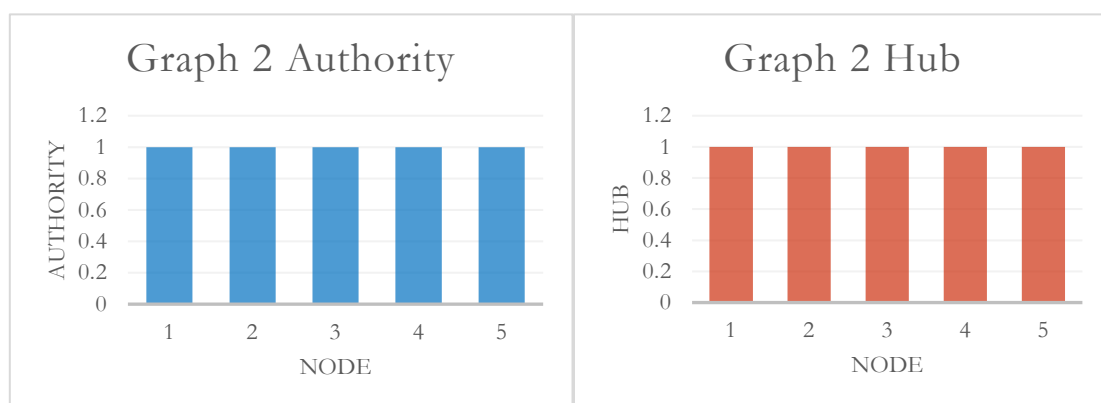
每個程式碼中皆以三個階段進行運作，首先進行設定參數與資料讀入矩陣的動作，接著第二階段帶入演算法進行計算，最後產生執行時間並把結果儲存為 CSV 檔。

### 三、 結果分析與討論

#### A. HITS 演算法



Graph 1 基本上屬於一個正常有順序連結情況，像是網路上填問卷或是註冊的時候需要一直點擊下一步的情況，可以簡單地透過增加節點 1 連出數以增加 Hub 值，或是將其他(2~5)節點連到節點 1 以增加節點 1 的 Authority。



Graph 2 為 Graph 1 連結 node 5 到 node 1 的結果，形成一個 circle 的圖形，除了用 Graph 1 的方法增加節點 1 的 Authority 與 Hub 值之外，由於是一個迴圈的圖形，每個節點的改變都會影響到別的節點或是自己。

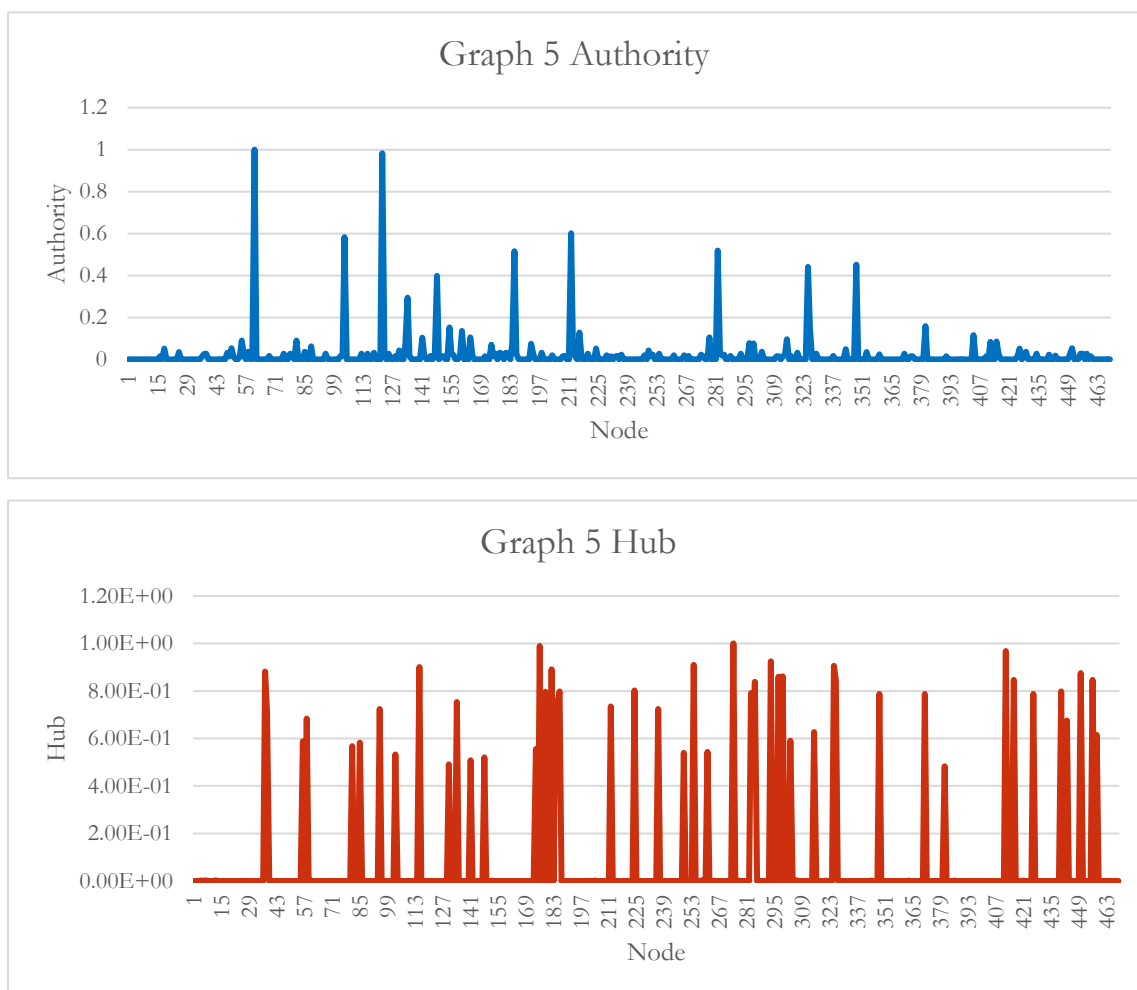


Graph 3 可以發現節點 2 與 3 是主要 Authority 與 Hub，增加節點 1 的 Authority

或 Hub 值，除了 Graph 1 的方法之外，由於節點 1 被連結的方式是由(4->3->2->1)，要增加節點 1 的 Authority 就必須增加節點 2 的 Hub 值、增加節點 3 的 Authority、增加節點 4 的 Hub 值(演算法迭代影響的關係)。

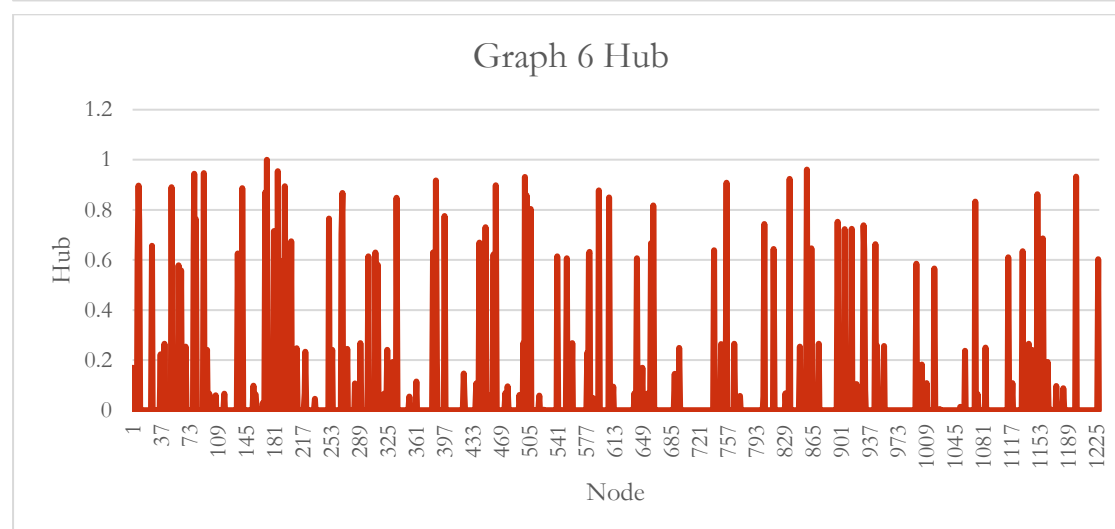
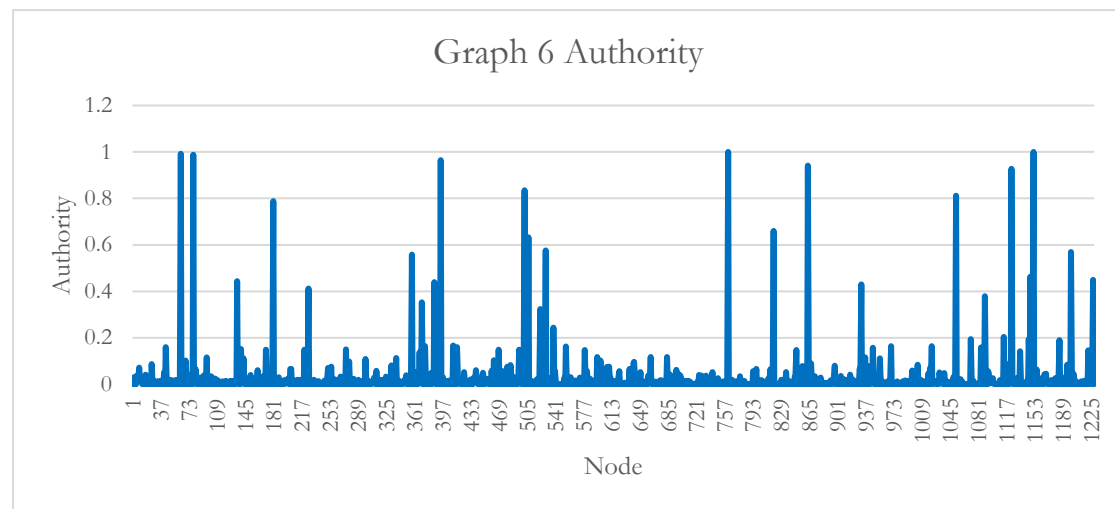


Graph 4 雖然節點不多，但是 edge 較多，連結的複雜度較高，可以觀察到 Authority 最高的節點集中在 1~5 之間，最高 Hub 為節點 1，顯示節點 1 是主要的入口網站，且使用者可能較常連到 1~5 的網頁。

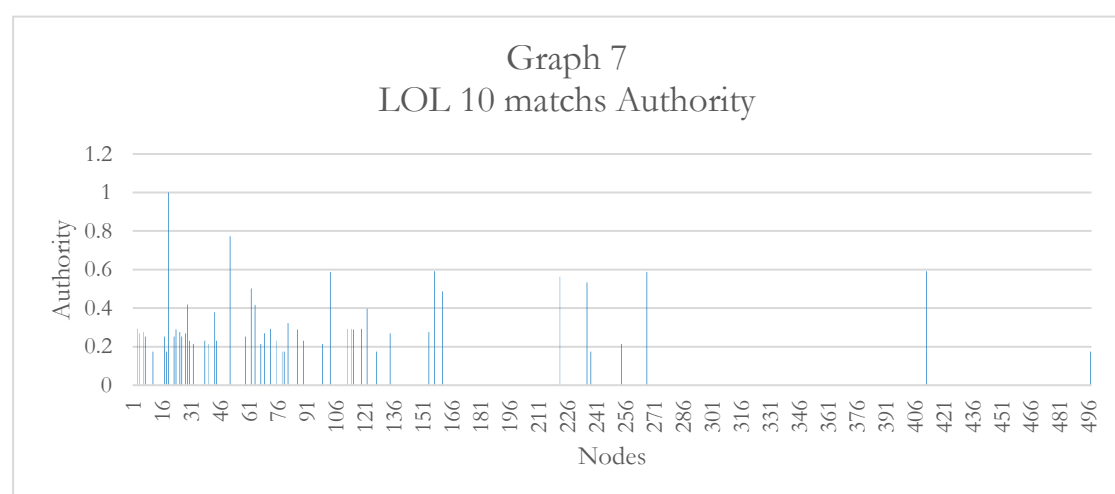


Graph 5 資料量較多，可以發現高 Authority 節點較少，但是高 Hub 節點卻比較多且平均，此現象類似於某些網站已經主導整個網路，大多數使用者皆會連到

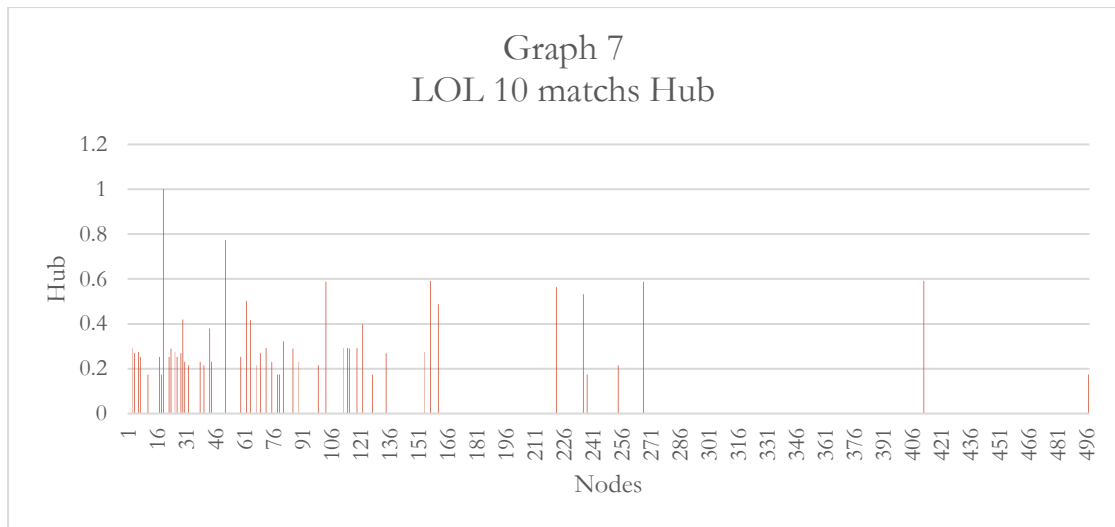
這些主要網站，不會去其他比較小的網站。在 Hub 方面，節點連出去的節點數量較多，大多數網站可能都會連到主要的網站，連到的網站可能就同樣那幾個，所以彼此之間的 Hub 值較平均。



Graph 6 的 edge 數量大約為 Graph 5 的 2 倍，可以看到 Graph 6 高 Hub 值的節點較多、密度較高。

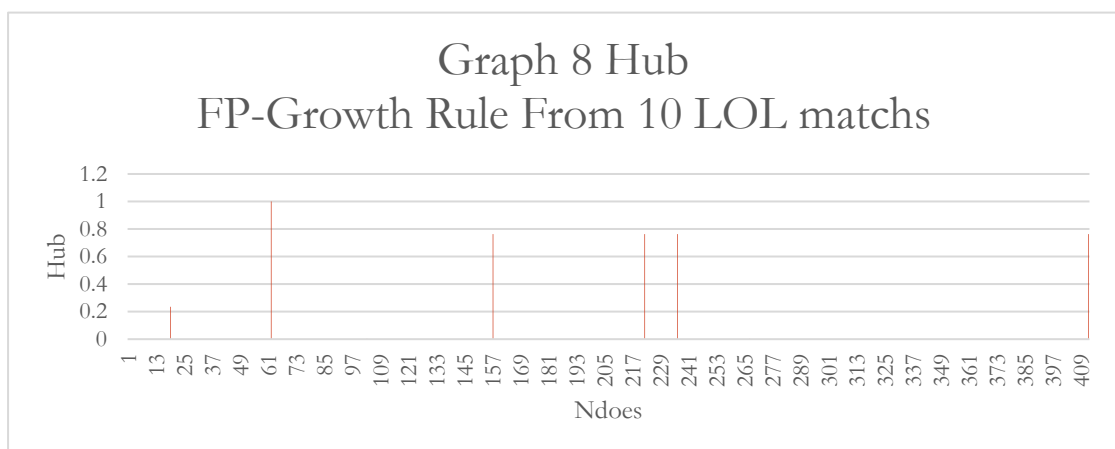
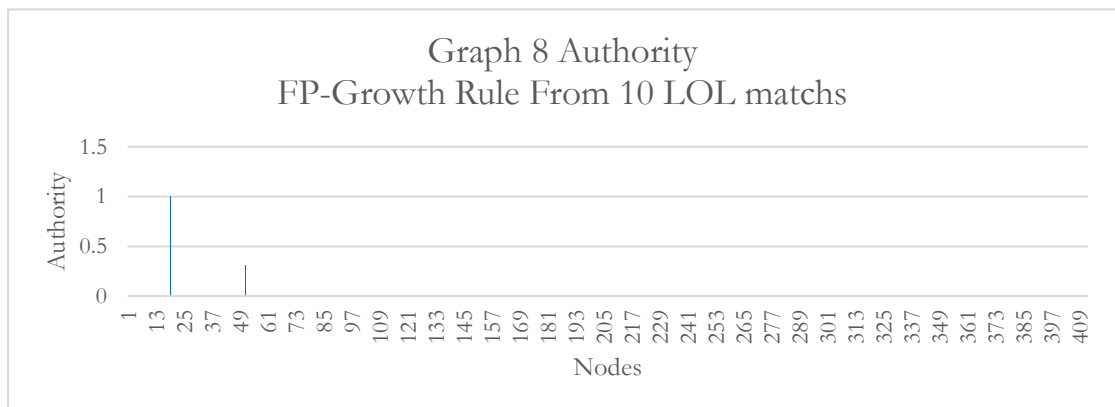






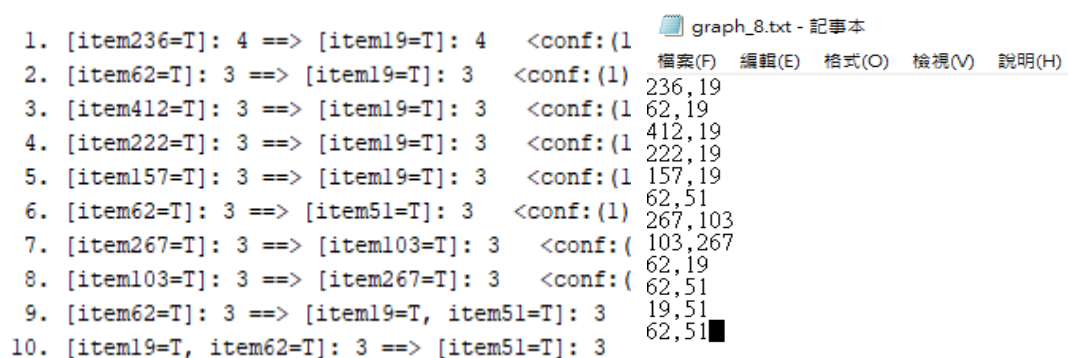
本專案使用英雄聯盟 10 場積分對戰(每場 10 隻腳色)的資料做為輸入，將每場比賽出現的 10 隻腳色 id 做 product 輸出然後以無向(bi-direct)的方式連結，最後輸出像 project3dataset 一樣的格式進行 HITS 分析的使用。

Graph 7 只需要看有值的部分，由於每個節點都是無向的，彼此互相連結，所以 Authority 與 Hub 兩個 graph 長的相同，數值較高代表較常被連到與連出去(每筆交易中較常出現的，像是節點 19)，較高的數值節點彼此之間有較大關聯(Frequent Item Set 較常出現)。



將十場 LOL 對戰資料輸入至 WEKA 進行 FP-Growth 分析，WEKA 輸出後

轉換成文字檔的形式，如下圖所示：



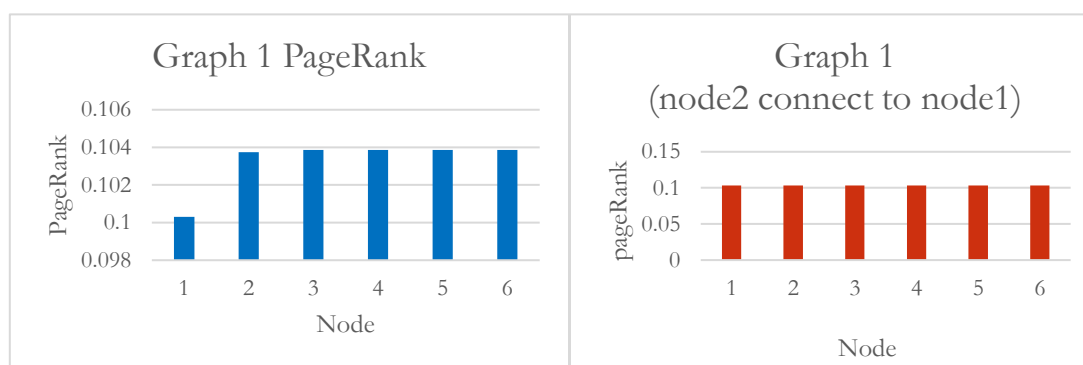
```

1. [item236=T]: 4 ==> [item19=T]: 4 <conf:(1 236,19)
2. [item62=T]: 3 ==> [item19=T]: 3 <conf:(1 62,19)
3. [item412=T]: 3 ==> [item19=T]: 3 <conf:(1 412,19)
4. [item222=T]: 3 ==> [item19=T]: 3 <conf:(1 222,19)
5. [item157=T]: 3 ==> [item19=T]: 3 <conf:(1 157,19)
6. [item62=T]: 3 ==> [item51=T]: 3 <conf:(1 62,51)
7. [item267=T]: 3 ==> [item103=T]: 3 <conf:(1 267,103)
8. [item103=T]: 3 ==> [item267=T]: 3 <conf:(1 103,267)
9. [item62=T]: 3 ==> [item19=T, item51=T]: 3 <conf:(1 62,19 62,51)
10. [item19=T, item62=T]: 3 ==> [item51=T]: 3 <conf:(1 19,51 62,51)

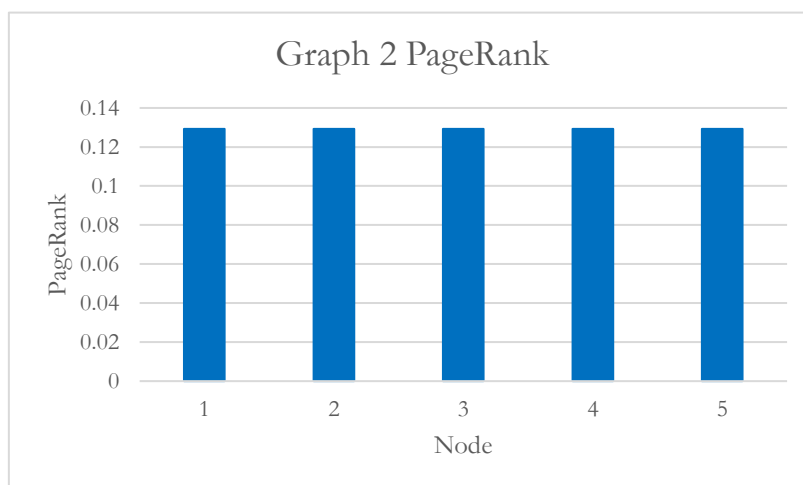
```

Graph 8 結果顯示 HITS 的 Authority 與 Hub 關係與關聯法被推導與推導的概念類似，在 Authority 值較高的節點為較常被關聯法則推導出來的遊戲腳色，至於 Hub 值較高的節點則為推導的遊戲腳色，對應 Graph 7 的峰值也是較常出現遊戲腳色。

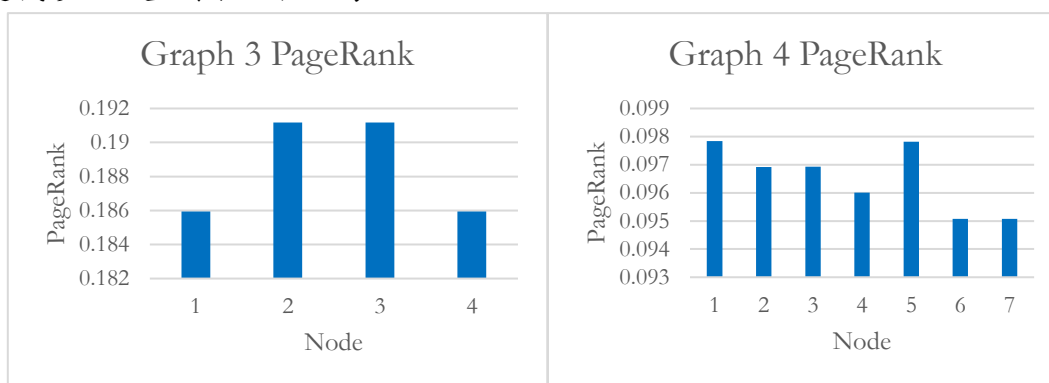
## B. PageRank 演算法



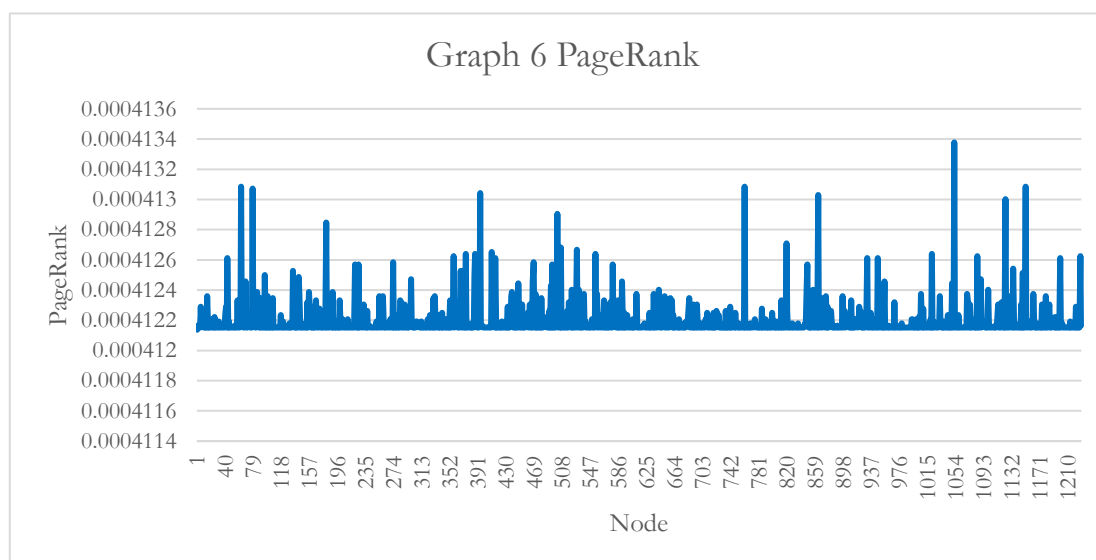
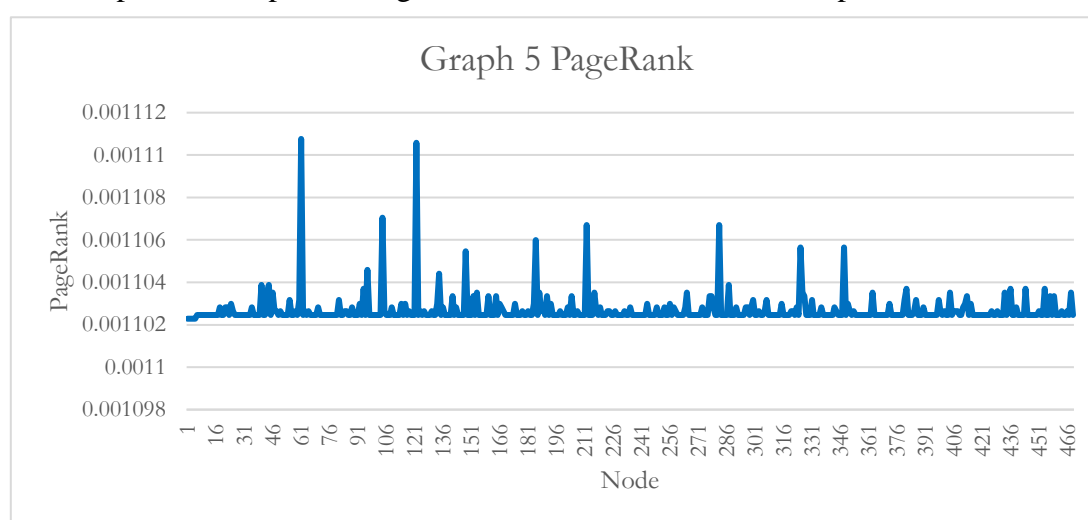
Graph 1 可以發現節點從連結方向開始收斂趨向平緩，PageRank 的值與連入的數量有關，要增加 node 1 的值只需要讓每個節點都被連到一次即可，如右上圖額外連節點 2 到節點 1 即可。



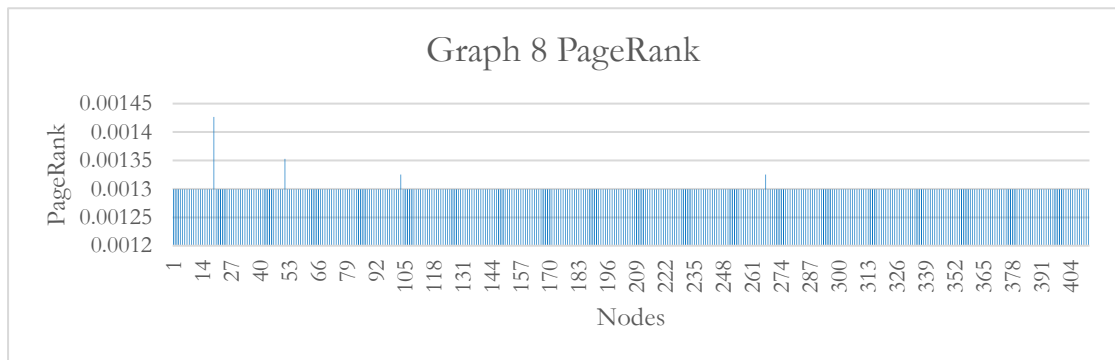
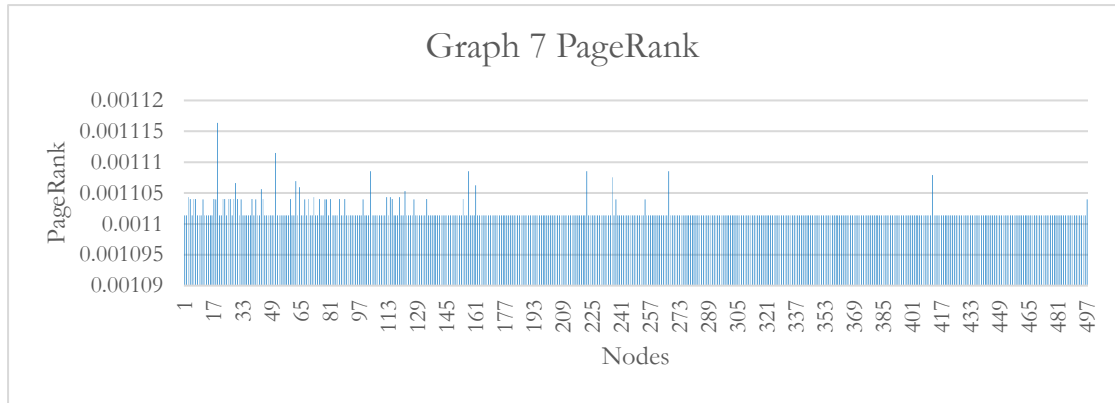
Graph 2 所有值都相同，PageRank 一開始先平均所有節點的值，然後不斷迭代修正，直到小於容錯為止。



Graph 3 和 Graph4 的 PageRank 結果與 HITS 產生的 Graph 相同。



Graph 5 與 Graph 6 的結果與 HITS 的 Hub 結果差不多，Graph 6 的密度也同樣比 Graph5 高。



Graph 7 與 Graph 8 的結果也與 HITS 的 hub 分析結果相似。

### C. SimRank 演算法

1	0	0	0	0	0
0	1	0	0	0	0
0	0	1	0	0	0
0	0	0	1	0	0
0	0	0	0	1	0
0	0	0	0	0	1

Graph 1 SimRank

1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	0	1	0
0	0	0	0	1

Graph 2 SimRank

Graph 1 和 Graph 2 因為沒有兩個以上的節點同時連入某節點的情況，無法分析之間的關聯性，所以皆為無意義的對稱矩陣。

1	0	0.052632	0
0	1	0	0.052632
0.052632	0	1	0
0	0.052632	0	1

Graph 3 SimRank

Graph 3 的節點 4 與節點 2 都連到了節點 3，所以從矩陣可看出節點 2 與 4 有關聯，節點 4 與節點 2 也有關聯，其關聯度相等。

1	0.010386	0.010074	0.013663	0.008138	0.001364	0.025962
0.010386	1	0.023462	0.017731	0.018898	0.034125	0.001338
0.010074	0.023462	1	0.034059	0.018371	0.03406	0.034059
0.013663	0.017731	0.034059	1	0.013625	0.050407	0.050407
0.008138	0.018898	0.018371	0.013625	1	0.026025	0.001225
0.001364	0.034125	0.03406	0.050407	0.026025	1	0.000814
0.025962	0.001338	0.034059	0.050407	0.001225	0.000814	1

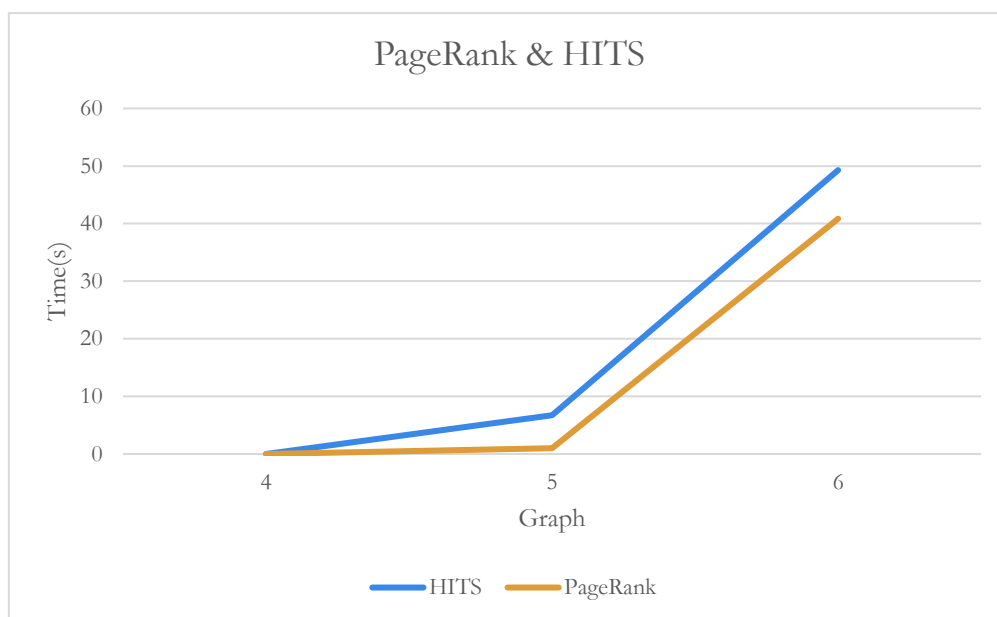
Graph 4 SimRank

Graph 4 可觀察出節點 4 與節點 7，以及節點 4 與節點 6 有較大的關聯度。

Graph 5 和 Graph6 由於矩陣太大，無法顯示出來，其結果存放於輸出結果資料夾當中。

#### 四、 運算效能分析

本專案在相同的容錯值的前提下，分別測試 HITS 與 PageRank 的計算速度，基本上 PageRank 與 HITS 時間複雜度差異不大，執行結果也差不多，但 PageRank 相較於 HITS 有較快的速度。



## 五、 結論

在這次的專案實作後，我認為可以加入一些文本分析的工具到 Link Analysis 當中，像是 TF-IDF 之類的，其功用可以分析網頁中的內容有哪些是有關聯的，再利用分群演算法進行分群，網頁彼此之間的關係能夠更加明確。

在這三個演算法中，我認為 HITS 相較於 PageRank 較有參考價值，雖然 PageRank 執行比 HITS 有效率，但只有看輸入而已；而 HITS 能夠得到較多節點與節點之間的資訊，如果再結合用於分析網頁之間的關係的 SimRank，對於整體分析將會更加完整。

然而，分析結果得到的這些值只能夠當作參考，網路上充斥著一堆假網站與無意義的連結，像是釣魚廣告網站，此外 Link Analysis 並沒有考慮到網頁本身的內容、特性等等，像是購物網站常常會連到評價論壇之類的，連結分析只能夠分析使用者瀏覽網頁的行為過程，但對於要充分判斷之間的關係依然相當耗費人力、成本與時間，現今的技術也不夠成熟，這些演算法也只能夠當作最初的參考，其餘還是必須交由專業人員來判斷。