資料探勘 HW3

說明：

本次作業學習實作 HITS, PageRank, SimRank 這三種演算法，並計算圖中各 vertex 的 authority, hub, PageRank, SimRank 值找出不同演算法各自覺得重要的 vertex 與發現各演算法的特點。

報告架構：

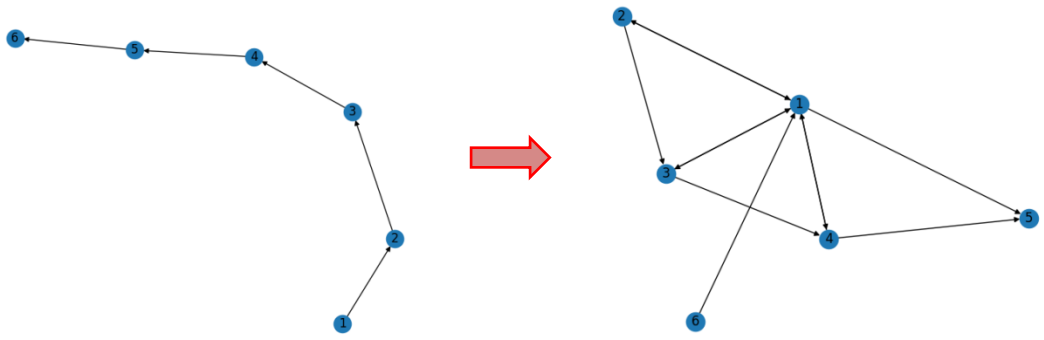
- ✓ Find a way
- ✓ Algorithm description
- ✓ Result analysis and discussion
- ✓ Effectiveness analysis
- ✓ Conclusion

Find a way

若要增加 vertex 1 的 authority, hub and PageRank 值，只需增加 vertex 1 的 child and parent 數量即可。若是以刪除其他 vertex 的 edge 作為考量，只需刪除和 vertex1 無關的 edge 即可。

增加關於 vertex1 的 edge 很容易想像可以提高 authority, hub and PageRank 值，刪除的話則是由於計算 authority, hub and PageRank 時會需要做正規化的動作，因此只要其他與 vertex1 無關的 edge 減少，vertex1 的 authority, hub and PageRank 值也會相對提高。

- **Graph1 為例：**



說明：

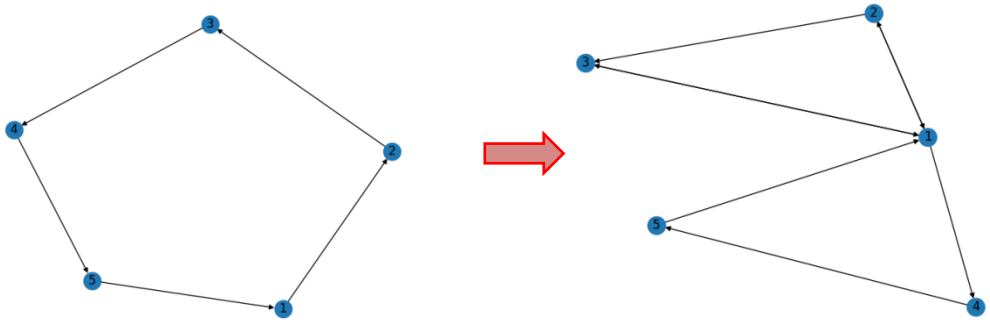
左圖是 graph1 原本的圖，右圖是經過增加/刪除 edge 的圖。

在原本的 graph1 中，增加了(1,3), (1,4), (1,5), (2,1), (3,1), (4,1),(6,1)的 edge，並刪除了(5,6)的 edge，使 vertex1 的 authority, hub, PageRank 值都有上升，但由於 vertex1 authority, hub, PageRank 值都上升，相對其他有某些 vertex 的值會下降。

例如：vertex 5 的 hub 值下降, vertex 6 的 authority 值下降。

	1	2	3	4	5	6
Authority (Before)	0.000	0.200	0.200	0.200	0.200	0.200
Authority (After)	**0.289**	0.117	0.198	0.198	0.198	0.000
Hub (Before)	0.200	0.200	0.200	0.200	0.200	0.000
Hub (After)	**0.289**	0.198	0.198	0.198	0.000	0.117
PageRank (Before)	0.056	0.107	0.152	0.193	0.230	0.263
PageRank (After)	**0.296**	0.113	0.163	0.186	0.196	0.046

- **Graph2 為例：**



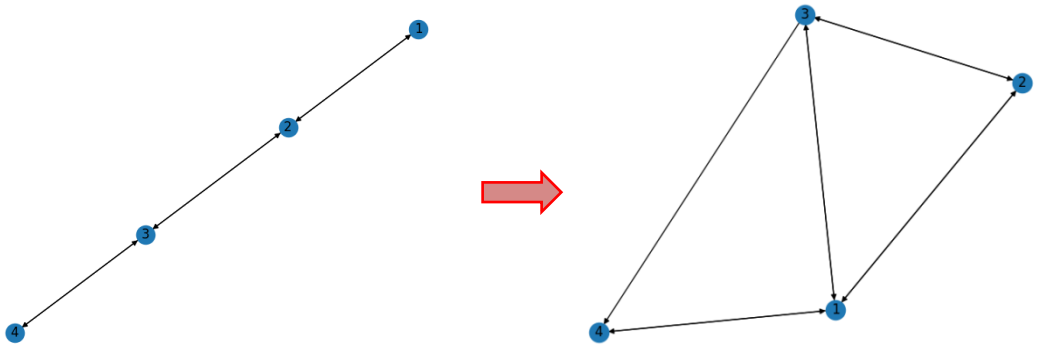
說明：

左圖是 graph2 原本的圖，右圖是經過增加/刪除 edge 的圖。

在原本的 graph2 中，增加了(1,3), (1,4), (2,1), (3,1)的 edge，並刪除了(3,4)的 edge，使 vertex1 的 authority, hub, PageRank 值都有上升。

	1	2	3	4	5
Authority (Before)	0.200	0.200	0.200	0.200	0.200
Authority (After)	**0.333**	0.167	0.333	0.167	0.000
Hub (Before)	0.200	0.200	0.200	0.200	0.200
Hub (After)	**0.333**	0.333	0.167	0.000	0.167
PageRank (Before)	0.200	0.200	0.200	0.200	0.200
PageRank (After)	**0.387**	0.136	0.198	0.136	0.143

- Graph3 為例：



說明：

左圖是 graph3 原本的圖，右圖是經過增加/刪除 edge 的圖。

在原本的 graph2 中，增加了(1,3), (1,4), (3,1), (4,1)的 edge，並刪除了(4,3)的 edge，使 vertex1 的 authority, hub, PageRank 值都有上升。

	1	2	3	4
Authority (Before)	0.191	0.309	0.309	0.191
Authority (After)	**0.264**	0.264	0.209	0.264
Hub (Before)	0.191	0.309	0.309	0.191
Hub (After)	**0.325**	0.209	0.350	0.117
PageRank (Before)	0.172	0.328	0.328	0.172
PageRank (After)	**0.367**	0.203	0.227	0.203

Algorithm description

以下分別介紹 HITS, PageRank, SimRank 演算法實作 code 流程。

✧ Hyperlink-Induced Topic Search algorithm (HITS)

```
def HITS(graph, max_iteration, threshold):
```

設定引入三個 argments 分別為 graph, max_iteration, threshold。

Graph: 引入要計算各點 authority and hub 的圖。
(以字典輸入，字典的 key 為圖中的 vertex, value 為為 key 的 vertex 有射入哪些 vertex 的邊)
Max_iteration: 最大 iteration 次數，在本次的實作中設為 30。
Threshold: 當各 vertex 前一次的 authority 值與本次的相減 加上 各 vertex 前一次的 hub 值與本次的相減，令此值為 ε，若 ε 的值小於設定的 threshold，則結束此 function，並 return 各 vertex 的 authority 值與 hub 值。
在本次的實作中設為 10^{-6}。

STEP1：
初始化 authority 與 hubs，將所有 vertex 的 authority 與 hub 初始化為 1

```
def initialization_HITS(node_list):

    # Build dictionary
    authority = hub = {i: 1 for i in node_list}
    return authority, hub
```

STEP2:
設定共要跑 max_iteration 次，並先記錄上一回各 vertex 的 authority 值與 hub 值，並更新各 vertex 的 authority 值與 hub 值，更新完後做正規化。

```
for i in range(max_iteration):
        if i == 0:
            pre_auth, pre_hub = authority_dic.copy(), hub_dic.copy()
```

```
        else:
            pre_auth, pre_hub = norm_auth.copy(), norm_hub.copy()

        authority_dic = update_authority(graph, node_list, pre_hub)
        hub_dic = update_hub(graph, node_list, pre_auth)

        norm_auth = normalization(authority_dic)
        norm_hub = normalization(hub_dic)
```

更新 authority 使用 update_authority() 這個 function

更新方式：

1. 先找出有哪些 vertices 射入要更新 authority 值的 vertex，令這些 vertices 為 set(A)
2. 計算 set(A)中所有 vertex 的 hub 值並相加，即為欲更新 authority 值的 vertex 新的 authority 值。

```
def update_authority(graph, node_list, hub_dic):

    new_auth = {i: sum(hub_dic[k]  for k, val in graph.items()  if i in val)  for i in node_list}
    return new_auth
```

更新 hub 使用 update_hub() 這個 function

更新方式：

1. 先找出要更新 hub 值的 vertex 射入哪些 vertices，令這些 vertices 為 set(B)
2. 計算 set(B)中所有 vertex 的 authority 值並相加，即為欲更新 hub 值的 vertex 新的 hub 值。

```
def update_hub(graph, node_list, auth_dic):

    new_hub = {i: sum(auth_dic[h]  for h in graph.get(i, []))  for i in node_list}
    return new_hub
```

最後使用 normalization() 這個 function 將更新好各 vertex 的 authority 與 hub 值做正規化，這邊採取用 1-norm 做正規化。

```
def normalization(dic):

    denominator = sum(dic.values())
    norm_dic = {k: dic[k] / denominator    for k in dic.keys()}
    return norm_dic
```

STEP3:

各 vertex 前一次的 authority 值與本次的相減 加上 各 vertex 前一次的 hub 值與本次的相減，令此值為 check_sum，若 check_sum 的值小於設定的 threshold，則結束此 function，並儲存成指定格式並 return 各 vertex 的 authority 值與 hub 值。

若未小於 threshold 則持續運作程式，直到設定的 max_iteration，則結束此 function，並儲存成指定格式並 return 各 vertex 的 authority 值與 hub 值。

```
        check_sum = 0

        if i >= 1:

            for k in authority_dic.keys():
                check_sum += (abs(pre_auth[k] - norm_auth[k]) + abs(pre_hub[k] - norm_hub[k]))

            if check_sum < threshold:

                output_norm_auth = ['%.3f' % norm_auth[k] for k in node_list]
                output_norm_hub = ['%.3f' % norm_hub[k] for k in node_list]

                return output_norm_auth, output_norm_hub

    output_norm_auth = ['%.3f' % norm_auth[k] for k in node_list]
    output_norm_hub = ['%.3f' % norm_hub[k] for k in node_list]

    return output_norm_auth, output_norm_hub
```

✧ PageRank Algorithm

$$PR(P_i) = \frac{(d)}{n} + (1-d) \times \sum_{l_{j,i} \in E} PR(P_j) / \text{Outdegree}(P_j)$$

```
def PageRank(graph, damping_factor, iteration):
```

設定引入三個 argments 分別為 graph, damping_factor, iteration。

Graph: 引入要計算各點 authority and hub 的圖。
(以字典輸入，字典的 key 為圖中的 vertex, value 為為 key 的 vertex 有射入哪些 vertex 的邊)
damping_factor: 阻尼係數，用於計算使用者點擊下一個頁面的機率。
本次實作中設為 0.15。
iteration: 最大 iteration 次數，在本次的實作中設為 30。

STEP1:
將所有 vertex 的 PageRank 值初始化為 1/N，其中 N 為 vertex 總數。並先建立 parent dictionary，預先記錄各 vertex 分別有哪些 vertices 射入。

```
    Page_Rank = initialization_PageRank(node_list)

    # Parent dic
    dic_parent = build_parent_dic(graph,node_list)
```

STEP2:
設定共會跑多少次 iteration，並依序更新各 vertex 的 PageRank 值。

```
    for i in range(iteration):
        for key in Page_Rank.keys():
            # Record node in
            parent_list = find_parent(key,dic_parent)
            parent_score = 0
            if len(parent_list) != 0:
                for p in parent_list:
                    parent_score += (old_pagerank[p] / len(graph[p]))
            Page_Rank[key] = damping_factor/total_pages + (1-damping_factor) * parent_score
```

首先使用 find_parent() 這個 function 找出欲更新 PageRank 的 vertex 的 parent vertices 有哪些。

```
def find_parent(child,parent_dic):
    return parent_dic[child]
```

接著找出每個 parent 的 PageRank 分數並將此分數除以該 parent 射到的 vertex 數量後，將所有 parent 的分數相加記錄在 parent_score 這個參數中。
最後計算(damping_factor / 總共的 vertex 數量) + (1-damping_factor) * parent_score 即是該 vertex 的 PageRank 分數。
計算完全部 vertex 的分數後做 normalization，這邊採用也是 1-norm，正規化後即可得到各 vertex 的 PageRank。

```
Page_Rank = normalization(Page_Rank)
```

每輪都要做正規化，直到跑完指定的 iteration。

✧ SimRank Algorithm

$$S(a,b)=\frac{C}{|I(a)||I(b)|}\sum_{i=1}^{|I(a)|}\sum_{j=1}^{|I(b)|}S(I_i(a),I_j(b))$$

```
def SimRank(graph, decay_factor, iteration):
```

設定引入三個 argments 分別為 graph, decay_factor, iteration。

Graph: 引入要計算各點 authority and hub 的圖。
(以字典輸入，字典的 key 為圖中的 vertex, value 為為 key 的 vertex 有射入哪些 vertex 的邊)
decay_factor: 阻尼係數，用於計算圖形間 vertex 相似度的精確度，通常設定在 0~1 的範圍內。若 decay factor 設定的值越高，則兩個 vertex 相似度受到的影響越小，反之則越大。
在本次的實作中 decay factor 設定為 0.7。
iteration: 最大 iteration 次數，在本次的實作中設為 30。

STEP1:

先初始化 SimRank matrix，根據 SimRank 的計算公式, Sim(a,b)，若 a = b 則 Similarity 值設為 1，並先建立 parent dictionary，預先記錄各 vertex 分別有哪些 vertices 射入，並也預先建立字典記錄 vertex 的 value 對應到 list 中的第幾項，之後要找出矩陣對應到的位置會較方便。

```
sim_matrix = np.eye(len(node_list))


# Parent dic
dic_parent = build_parent_dic(graph, node_list)

# Build index dic
index_dic = {}

for k in range(len(node_list)):
    index_dic[node_list[k]] = k
```

STEP2:

設定最多要跑幾個 iteration，並更新 SimRank Matrix 來將每個 vertex 的 SimRank 值做更新。

```
for itr in range(iteration):

    for i in range(len(node_list)):

        for j in range(len(node_list)):


            sim_matrix[i][j] = update_simrank(node_list[i],node_list[j],old_matrix,decay_factor,dic_parent,index_dic)
```

更新的方式是使用 update_simrank() 這個 function。
根據定義，首先先找到矩陣中要更新的值對應到的 a,b 兩個 vertex，並找出他們的 parent。如果 a, b 相同則直接 return 1，若其中有一個 vertex 沒有 parent, return 0。

若沒有提早 return，則計算兩個 vertex 的 similarity。計算方式則是將 a,b 兩個 vertex 的 parent 做比對，分別計算上一回 similarity 值並相加，最後將**相加過後的 similarity * decay factor / ((vertex a parent 個數)*(vertex b parent 個數))**即是。

將整個 SimRank 更新指定的 iteration 數後，即可得到各 vertex 之間最終的 similarity 值。

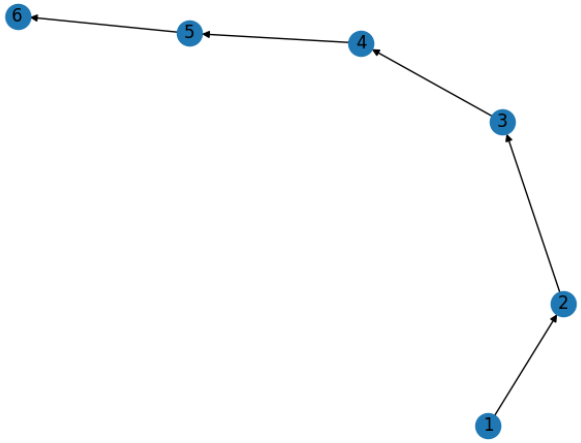
```
def update_simrank(a,b,old_sim_matrix,decay_factor,parent_dic,index_dic):
    if a == b:
        return 1

    else:
        in_a = find_parent(a,parent_dic)
        in_b = find_parent(b,parent_dic)

        if len(in_a) == 0 or len(in_b) == 0:
            return 0

        else:
            Sim_sum = 0
            for i in in_a:
                for j in in_b:
                    Sim_sum += get_sim_value(i,j,index_dic,old_sim_matrix)

        Sim_sum *= (decay_factor/(len(in_a)* len(in_b)))

    return Sim_sum
```

```
def get_sim_value(a, b, index_dic, old_sim_matrix):

    index_a = index_dic[a]
    index_b = index_dic[b]

    return old_sim_matrix[index_a][index_b]
```

Result analysis and discussion

Analysis for graph 1 to graph 3：

- ### Graph 1



HITS

由於 Graph 1 是單向傳遞的圖，可以看到 vertex 1 沒有任何 vertex 傳入，且 vertex 6 沒有傳出給其他的 vertex。故依照 authority and hub 的定義, vertex 1 的 authority = 0 (沒有其他任何 vertex 傳入), vertex 6 hub = 0 (沒有傳出給其他 vertex)

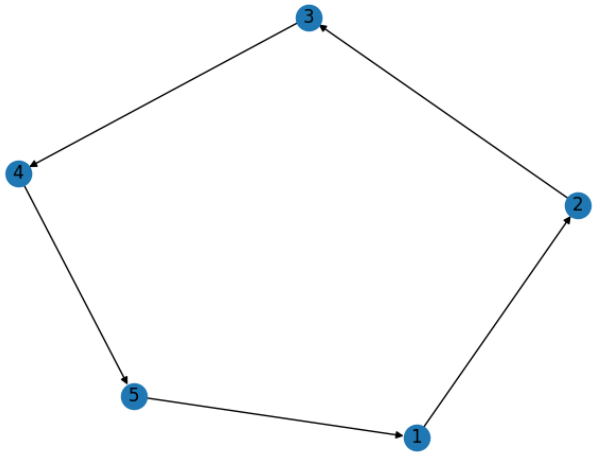
PageRank

由於 vertex1 沒有 vertex insert,因此在 PageRank 中, vertex1 的值會是最小的。

SimRank

由於沒有一個 vertex 有相同的 parent，故結果是單位矩陣，只有對角線的值為 1。

- Graph 2



HITS

由於 Graph 2 是環形傳遞的圖，可以看到每個 vertex 都只有一個 vertex 傳入，也只傳出至一個 vertex，故依照 authority and hub 的定義，所有的 vertex authority and hub 值都相同。

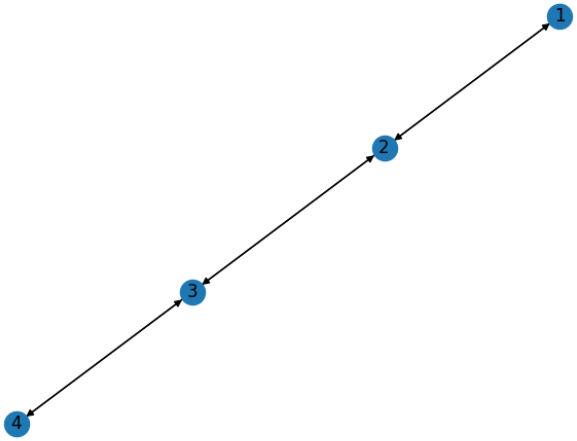
PageRank

由於所有 vertex insert 進來的 vertex 都只有一個，因此在 PageRank 中，所有 vertex 的值都是相同的。

SimRank

由於沒有一個 vertex 有相同的 parent，故結果是單位矩陣，只有對角線的值為 1。

- Graph 3



HITS

由於 Graph 3 中, vertex 2 and vertex 3 的 child and parent 數量相同, vertex 1 and vertex 4 child and parent 數量相同，因此, vertex 1 and vertex 4 的 authority and hub 值會相同，則 vertex 2 and vertex 3 的 authority and hub 值會相同，且 vertex 2 and vertex 3 的 authority and hub 值會較 vertex 1 and vertex 4 大，因為 vertex 2 and vertex 3 的 child and parent 數量較 vertex 1 and vertex 4 多。

PageRank

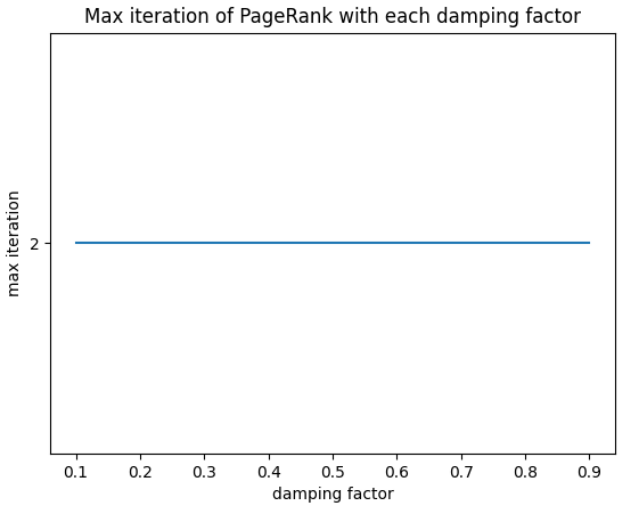
由於 vertex 2 and vertex 3 的 parent 數量較多，且他們的 parent 性質也都相同，因此 vertex 2 and vertex 3 有相同的 PageRank。同理在 vertex 1 and vertex 4，但由於 vertex 1 and vertex 4 的 parent 數量較 vertex 2 and vertex 3 少，且此圖較為單純，因此 vertex 1 and vertex 4 的 PageRank 值較低。

SimRank

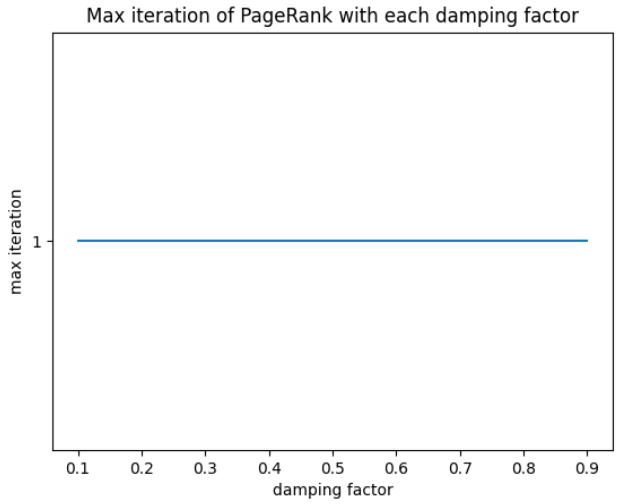
由於 SimRank 是看各個 vertex 的 parent 的相似度，而非 parent 的數量，因此雖然在 HITS 演算法中, vertex 1 and vertex 4 的 authority and hub 相同，但在 SimRank 中, vertex 1 and vertex 3 的 parents 中有共同的 parent，因此在 SimRank 中 vertex 1 and vertex 3 有一定的相似度，反而 vertex 1 and vertex 4 沒有相似，同理在 vertex 2 and vertex 4，在 SimRank 中也是有一定的相似度。

PageRank with different damping factor in PageRank：

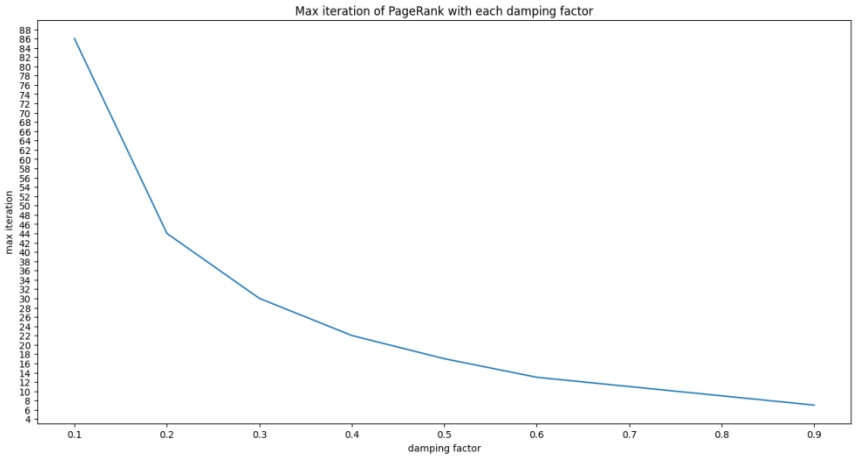
- Graph 1



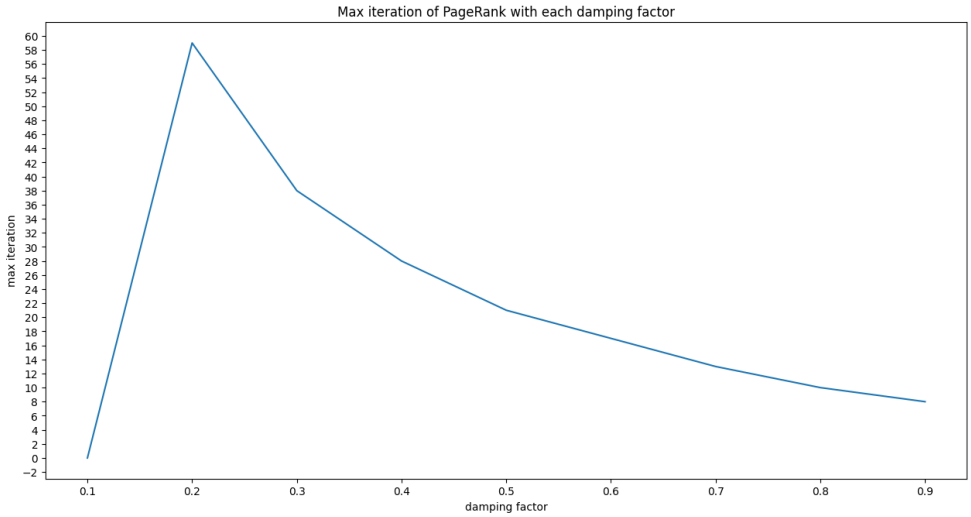
- Graph 2



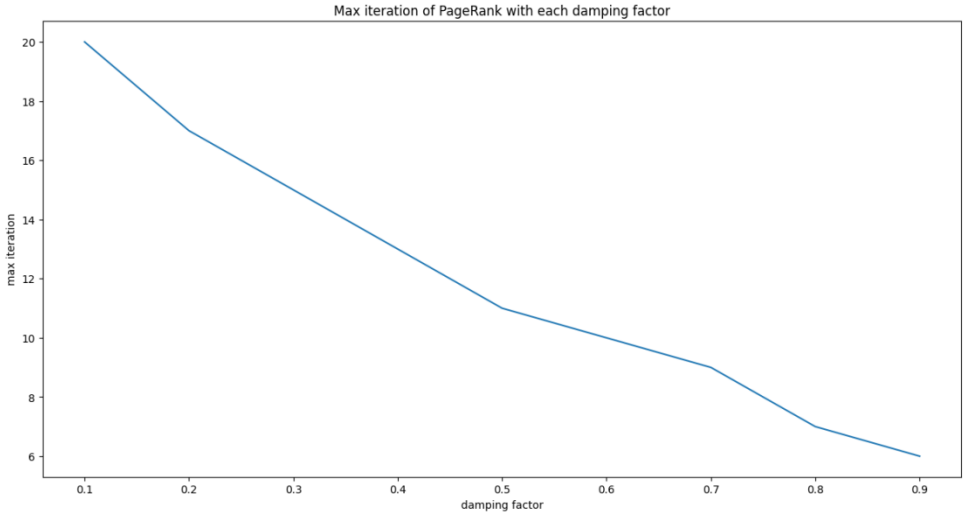
- Graph 3



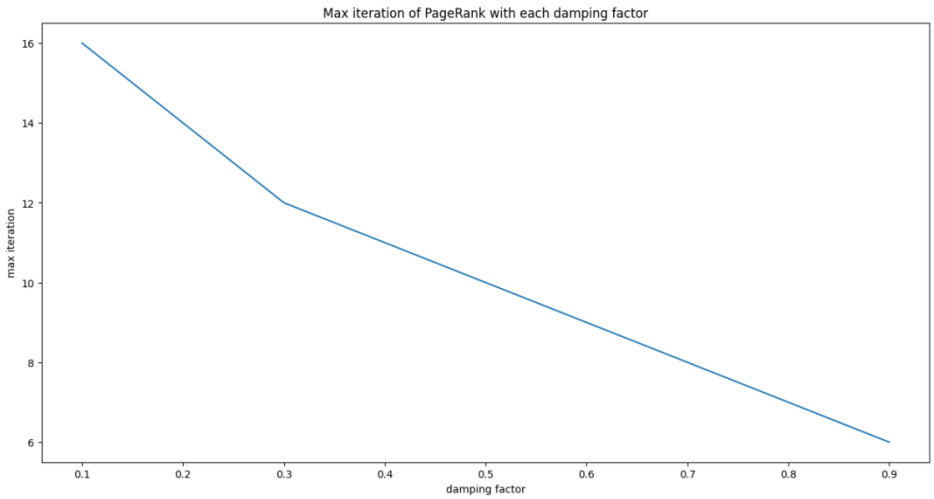
- ## Graph 4



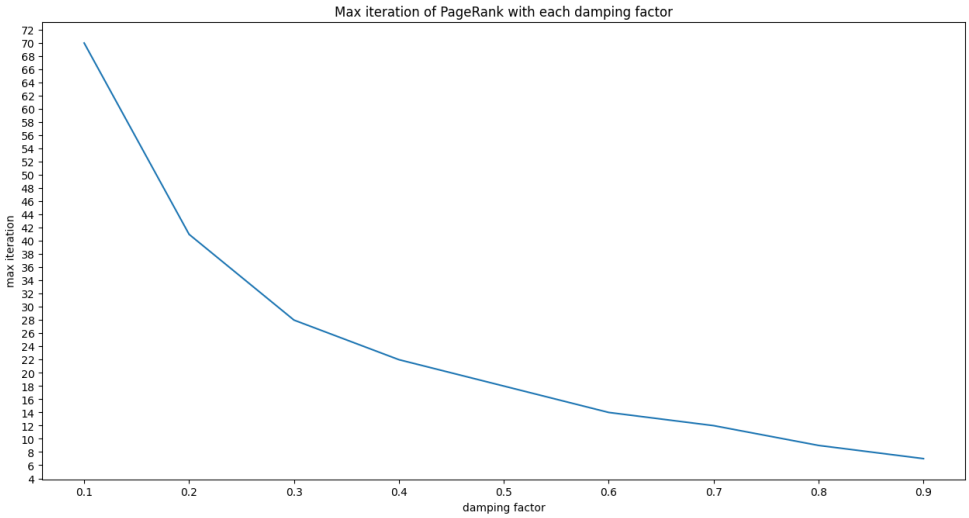
- ## Graph 5



- ## Graph 6



- ## Ibm-5000



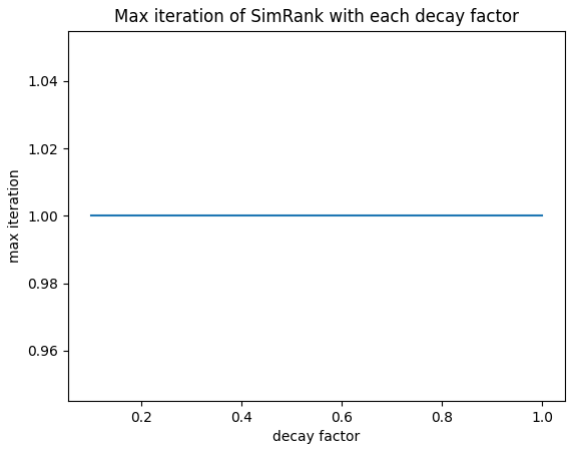
分析：

由 Graph 1~6 + ibm5000 對於不同的 damping factor 需要的 iteration 直到收斂的圖可知，由於 graph 1 and graph 2 的圖形較為簡單，且點 vertex and edge 數量不多，每個 vertex 也只有一個 parent，因此不管 damping factor 多少，很快就到收斂的狀態。

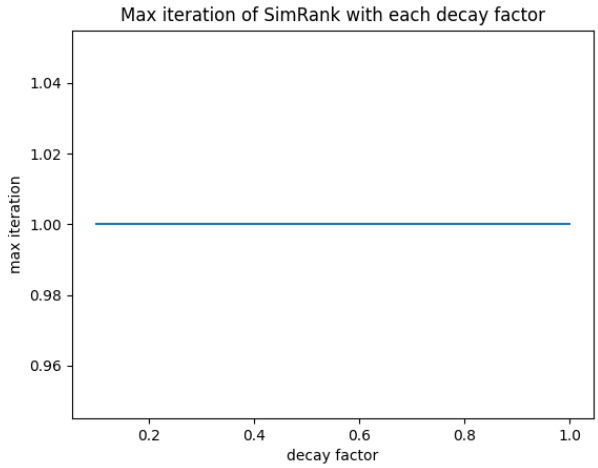
而 graph 3~6 + ibm-5000 基本上則是 damping factor 越低，需要的 iteration 數量要越多才會達到收斂，原因為 damping factor 越低，則自己的基本分數越低，分數比重較多由其他的 parent vertex 去計算，因此在計算 parent score 每次遞迴的過程中，分數的變化都會不小，因此要花較多次 iteration 才會達到收斂。Damping factor 越大，則分數大多數是來自自己的基本分，與最一開始初始值 1/n(n 表示全部的 vertex 數量)不會相差太大，因此較快達到收斂。

PageRank with different decay factor in SimRank：

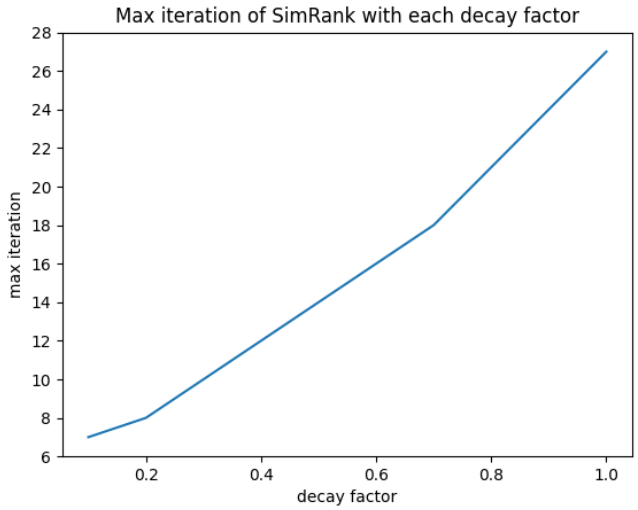
- ## Graph 1



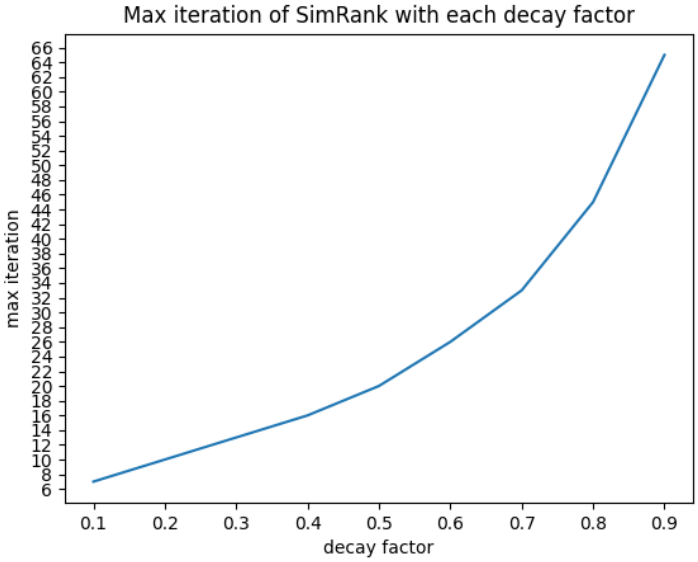
- Graph 2



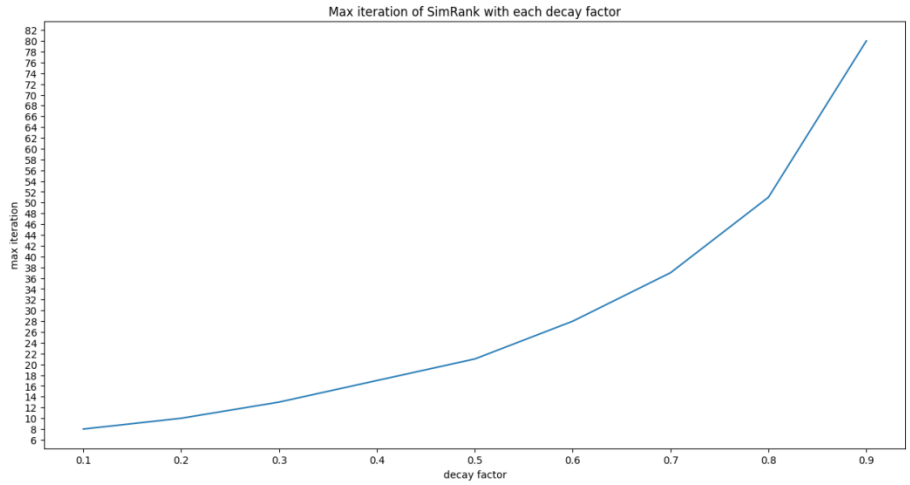
- Graph 3



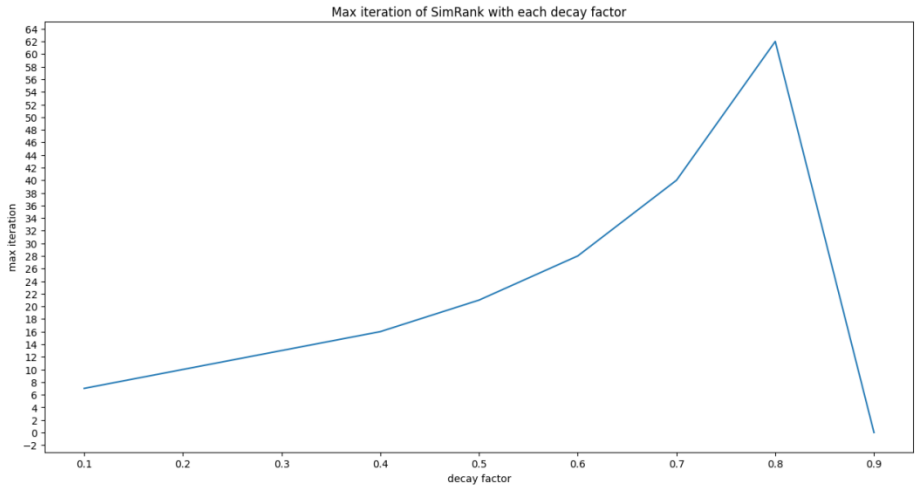
- Graph 4



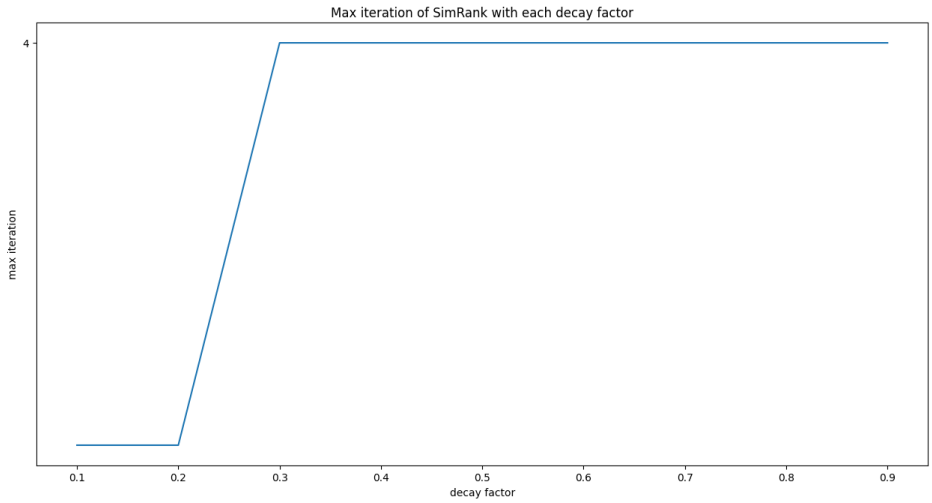
- ## Graph 5



- ## Graph 6



- ## Ibm-5000



分析：

由 Graph 1~6 + ibm5000 對於不同的 decay factor 需要的 iteration 直到收斂的圖可知，因為 graph 1-2 的圖形較為簡單，且沒有一個 vertex 有相同的 parent，故 SimRank Matrix 呈現一個單位矩陣的狀態，因此在很小的 iteration 情況下就直接達到收斂。

而 graph 3-6 圖形較複雜，vertex and edge 數量較多，隨著 decay factor 增加，需要越多的 iteration 才會達到收斂，由於 decay factor 越大則越會著重在各 vertex 的相似度上面，因此只要有 vertex and vertex 之間相似，則會要需要較多次的計算才可以達到收斂，因此有這樣的結果是可以預想的。

比較特別的狀況是在 ibm-5000 的圖形上面，ibm-5000 也是一個 vertex and edge 數量較多的圖形，與 graph 1-2 相比也是較為複雜，但卻不像 graph3-6 一樣越高的 decay factor 需要的 iteration 次數越多才會達到收斂，反而都是在 iteration = 4 時很快達到收斂，大概是因為在 ibm-5000，vertex 之間的共同的 parent 數量太少了，甚至大部分的 vertex 都沒有共同的 parent，造成 ibm-5000 的 SimRank Matrix 是稀疏矩陣，因此不需要太多的 iteration 即可使 SimRank Matrix 達到收斂，結果反而和 graph 1-2 很接近，可以呈現出 SimRank 的定義。

Effectiveness analysis

Algorithm Executed time (Sec)：

	HITS	PageRank	SimRank
Graph 1	0.0000	0.0000	0.0010
Graph 2	0.0000	0.0000	0.0010
Graph 3	0.0000	0.0000	0.0010
Graph 4	0.0000	0.0000	0.0040
Graph 5	0.3800	0.0120	17.7940
Graph 6	0.7130	0.0470	322.1885
Ibm-5000	0.3780	0.0270	224.4738

整體分析：

從表記錄的時間可知, graph1 到 graph 6 程式執行時間是逐步增加的，推測的原因為 graph 5-6 的 edge or vertex 數量較多，因此才會需要較多的時間。

以下試著實測若固定 vertex 或 edge 數量所需時間進行分析。

若固定 edge 數量 = 1000：

	HITS	PageRank	SimRank
Vertex = 50	0.0030	0.0040	10.1190
Vertex = 495	1.2970	0.0150	19.0390
Vertex = 735	9.9130	0.0320	23.0900

若固定 vertex 數量 = 100：

	HITS	PageRank	SimRank
Edge = 500	0.0011	0.0040	2.9050
Edge = 800	0.0080	0.0040	6.8980
Edge = 1100	0.0070	0.0050	12.772

由上面實驗結果可知不論固定 edge 數量增加 vertices, 或是固定 vertex 數量增加 edges, 只要 vertex or edge 數量增加, 則三個演算法運行的時間都會增加, graph1-6 + ibm-5000 中, 最為複雜的圖形是 graph 6, vertex 數量最多, edge 數量也最多, 因此運行的時間最長。

三個演算法分析：

從以上表格呈現數據來看，三個演算法執行時間為 SimRank > HITS > PageRank，分析原因為 SimRank 要比較各 vertex 的 parent 相似度，但一個 vertex 中可能有許多 parent，倆倆比較並計算分數的計算量非常大，加上一個 vertex 又會需要計算這個 vertex 與其他 vertex 的相似度，因此所需花費的時間會是最多。

而花費第二多的時間為 HITS 演算法，推測可能是 HITS 演算法除了計算各 vertex 的 authority 外，還要計算 hub 值，而 PageRank 只要計算 parent 的分數再除以該 parent 的 child 有多少個即可。

Conclusion

這次的作業實作了 HITS, PageRank, SimRank 演算法，實作的過程中與分析也更了解三個演算法如何計算、不同之處與他們著重的點。

不過這三個演算法要如何運用在實際的 page 網路上面會是一個問題, PageRank 的效率最好相對起來問題小一些，但在課程上面提到的假的 link，或是自己一直指向自己相關的 Page 也還是一個問題，此外，若實際運用到 page 上面，單純以 link 來計算的話是沒有考慮到內容的，這也會是一個問題，或許也可以再思考看看可以如何解決或改善內容相關的問題。