DESIGN AND IMPLEMENTATION OF PAGE RANK ALGORITHM

BY

N IMAYAVALLI

Problem Definition:

- To design and implement the page rank algorithm, to rank a corpus of webpages by considering their inlinks and outlinks (including handling of spider traps and dead ends).
- To display a graph of some of the pages(nodes) and their links (edges), with their sizes proportional to their page rank values.

Algorithm used: Pseudo Code

Convergence check: Pseudo code

```
sum = 0
    if loop is 1:
        notconverged
    else:
        for all N:
            Sum = sum + difference between rold[i] and rnew[i]
        if sum > 0.5(the chosen value):
            notconverged
    converged
```

Initialize all rold elements to 1/N

Get the list of degrees of all nodes as a vector: Pseudo code

By traversing the edges

And increasing the value of the ith element in the list for every line in the file starting with the value I

Get the list of inlinks of all nodes as a vector: Pseudo code

By traversing the edges

And adding 1 to the number of inlinks for every line in the file ending with the value i

Pseudo code to traverse the edges

Open file

For all N nodes

Set l = 0

for line in f:

Increase l by 1

if l > N+1: (to skip all lines until we get to the edges)

lister = line.split() (to get the 2 nodes of each edge as a list of 2 elements)

Initialize the rank vector

Get degrees of all nodes

Repeat until convergence:

Update rold with rnew if loop is not the first loop

Calculate rnew2 for each node:

getinlinks

for node in inlinks:

```
                rnew2[i] += rold[node]/degrees[node]
        rnew2[i] = B * rnew2[i]
    #Accounting for spider traps and dead ends
    s = 0
    for j in range(0, N):
        s += rnew2[j]
    for j in range(0, N):
        rnew[j] = rnew2[j] + (1-s)/N
```

I have used rank vectors instead of rank matrices, which reduces the scale of memory used and time taken.

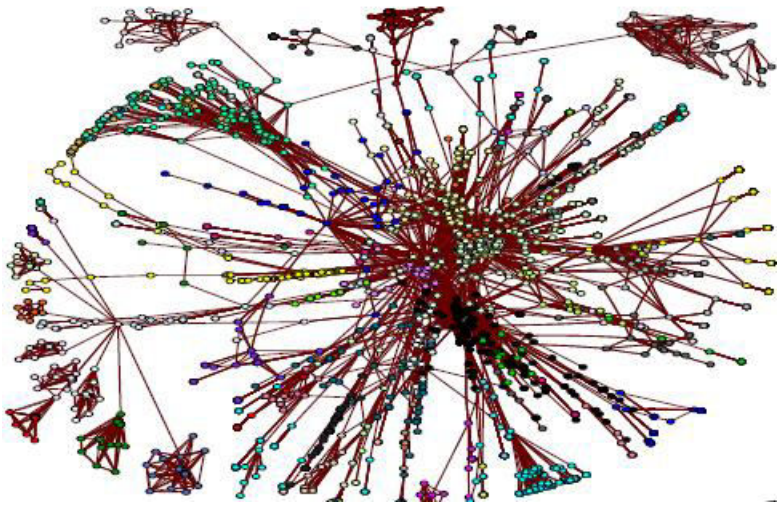
Dataset that I have used:

Kenneth Massey's Information Retrieval webpage:

Hollins.edu webbot crawl from January 2004

6012 nodes(webpages), 23875 edges(links)

To support the fact that this corpus has spider traps and dead ends and hence that they have been handled by the above algorithm, I have attached the visualization of the hollins.edu webbot crawl, as presented in another source - Finding Communities in Site Web-Graphs and Citation Graphs By Antonis Sidiropoulos from Data Engineering Lab, Department of Informatics, Aristotle University



Running time: Approximately 105s for the algo

Enhancements:

- Rank vectors eliminating the use of larger rank matrices -> Implementable on very large datasets and graphs
- Topic specific - page rank algo
- Multiple node graphs of different distributions of nodes
  - All 6012 nodes (pretty messy)
  - n popular nodes (default n value is set to 30)

    The 1st 10 nodes have only 6 nodes with edges in between them

    The 1st 20 nodes have 11 nodes sharing edges

    The 1st 30 nodes have 14 nodes sharing edges

    The 1st 50 nodes have 31 nodes sharing edges
  - 20 specific nodes

    All of them have at least 1 common edge
  - n random nodes (default n value is set to 200)

    Higher the value of n, higher is the probability of getting nodes with common edges. On avg, for n=200 to 250, 30-40 nodes will be displayed