Page Rank for graphs where nodes get added dynamically

(Rohith Parjanya,M20CS009)

Introduction:

Most real-life networks are dynamic in today's world, and finding page rank will help us analyze many things.

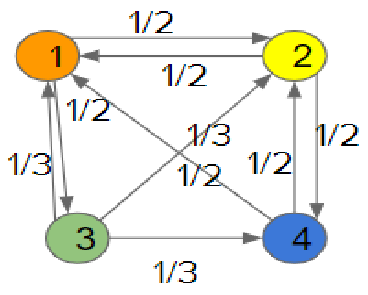
PageRank:

Ranking the web pages based on the number of backlinks that a page has and the quality of the backlinks. (or) the probability of reaching a node is treated to be page rank; the greater the probability, the higher its rank will be.

Process: (Larry page, 1999)

1. Let [p] be the 1 step transition probability matrix
2. Let π be the initial rank vector. Initially, we initialize this rank vector with equal probabilities, i.e., < 1/n, 1/n, ….. >, n here is the number of nodes of the graph
3. Iterate:
   a. $[P]^1 * \pi_0 = \pi_1$
   b. $[P]^2 * \pi_1 = \pi_2$
   c. $[P]^3 * \pi_2 = \pi_3$
   d. $\ldots\ldots\ldots\ldots\ldots\ldots$
   e. Stop if $\pi_{k-1} == \pi_k$
4. π_k Is the final rank vector.

Example:



Initial rank vector (π)= < ¼, ¼, ¼, ¼ >

1 [0, ½, ½, 0]

2 [½, 0 , 0, ½]   → $[P]^1$ matrix

3 [⅓, ⅓, 0, ⅓]

4 [½ , ½, 0, 0]

The idea behind Pagerank is to find the dominant eigenvector, which is derived by the above procedure.

PageRank formulation:

$$\tilde{P} = cP + (1-c)\frac{1}{n}E \rightarrow (1)$$

C here is the damping factor (The probability, at any step, that the person will continue is a **damping factor**). The recommended value for c is 0.85, P is transition probability matrix, E is a unit matrix of size n × n.

$$[P] * \pi = \pi \rightarrow (2)$$

The idea to arrive at randomized Monte Carlo algorithms:

From (1) and (2).

$$\boldsymbol{\pi} = \boldsymbol{\pi}\tilde{P} = \boldsymbol{\pi}cP + \frac{1-c}{n}\boldsymbol{\pi}E = \boldsymbol{\pi}cP + \frac{1-c}{n}\underline{\mathbf{1}}$$

$$\boldsymbol{\pi}(I - cP) = \frac{1-c}{n}\underline{\mathbf{1}}.$$

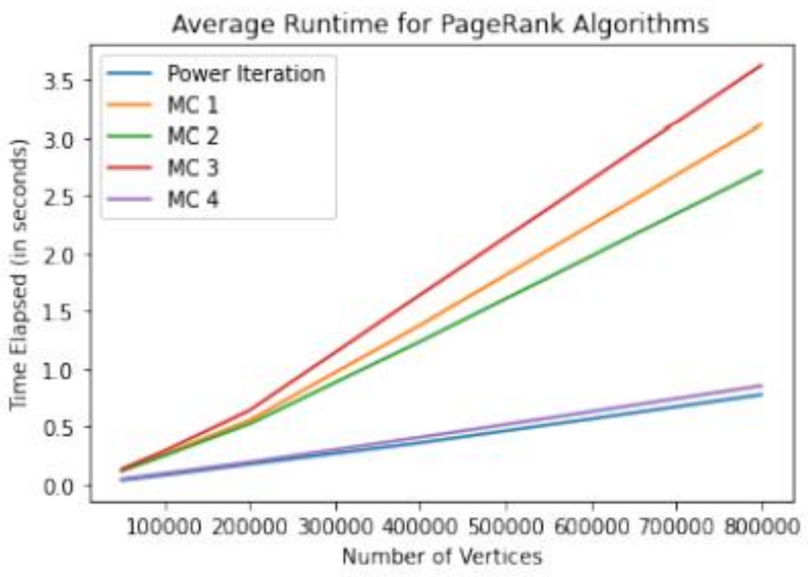
$$\boldsymbol{\pi} = \frac{1-c}{n}\underline{1}[I - cP]^{-1}$$

$$\pi_j = \frac{1-c}{n}\sum_{i=1}^{n}[I - cP]_{ij}^{-1} \rightarrow (3)$$

Consider a random walk that starts from a randomly chosen page. Assume that at each step, the random walk makes a transition according to the matrix P with probability c or terminates with probability (1 − c). Then it follows from (3) that the end-point of this random walk (the last visited page before the random walk terminates) appears to be a sample from the distribution π. Thus, after repeating the process many times, the estimate for π_j For j = 1...n can be determined as the number of times the random walk terminates at page j divided by the total number of random walks.

There are four types of randomized Monte Carlo algorithms:

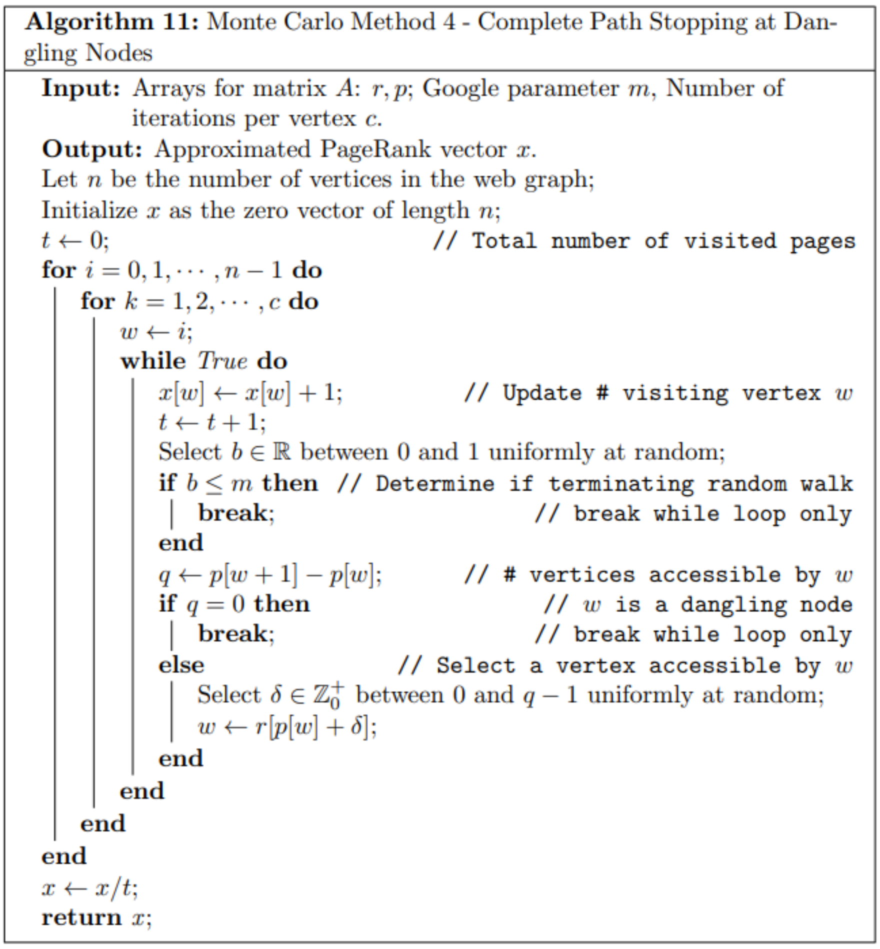
1. MC end-point with random start
2. MC end-point with cyclic start
3. MC complete path
4. **MC complete path stopping at dangling nodes**



Plot for the runtime of the various PageRank algorithms against the varying sizes of the web graphs. When comparing the runtimes between the four Monte Carlo methods and the Power Iteration, it turns out that the Power Iteration has faster performance overall. The fourth Monte Carlo method has a comparable runtime to the Power Iteration, but the output of the Monte Carlo method is not as accurate as of that of the Power Iteration.[exploring page rank algorithms, Vargas brian,2020]

MC complete path stopping at dangling nodes:

Simulate the random walk starting exactly m times from each page. Evaluate πj as the total number of visits to page j divided by the total number of visited pages. [bthesis]



Algorithm 11: Monte Carlo Method 4 - Complete Path Stopping at Dangling Nodes

```
Input: Arrays for matrix $A$: $r, p$; Google parameter $m$, Number of
       iterations per vertex $c$.
Output: Approximated PageRank vector $x$.
Let $n$ be the number of vertices in the web graph;
Initialize $x$ as the zero vector of length $n$;
$t \leftarrow 0$;                              // Total number of visited pages
for $i = 0, 1, \cdots, n-1$ do
    for $k = 1, 2, \cdots, c$ do
        $w \leftarrow i$;
        while True do
            $x[w] \leftarrow x[w] + 1$;             // Update # visiting vertex $w$
            $t \leftarrow t + 1$;
            Select $b \in \mathbb{R}$ between 0 and 1 uniformly at random;
            if $b \leq m$ then  // Determine if terminating random walk
                break;                          // break while loop only
            end
            $q \leftarrow p[w+1] - p[w]$;          // # vertices accessible by $w$
            if $q = 0$ then                         // $w$ is a dangling node
                break;                          // break while loop only
            else                         // Select a vertex accessible by $w$
                Select $\delta \in \mathbb{Z}_0^+$ between 0 and $q-1$ uniformly at random;
                $w \leftarrow r[p[w] + \delta]$;
            end
        end
    end
end
$x \leftarrow x/t$;
return $x$;
```

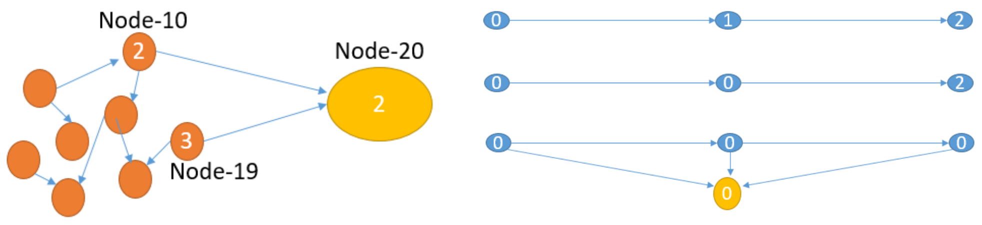
From the above Algorithm, we derive a new approach to calculate page rank for dynamically added nodes.

Dynamic Approach:

Terms to understand in Algorithm:

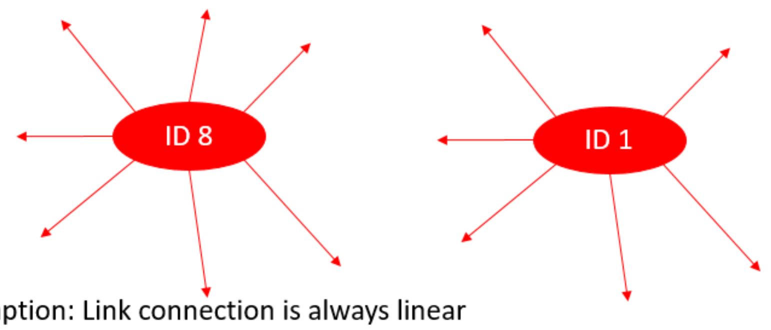
LinkID: The link ID of a node is the ID given to a node to identify the node or to identify to which link does this node belongs to. In a simple sense, it is used to keep track of all the links (distinct chains in the graph)

Note: Here, we follow the convection of assigning a lower linkID to the node if multiple links exist. This convention is not going to affect the result of the procedure, and this is opted just to keep things simpler for tracking linkIDs length.

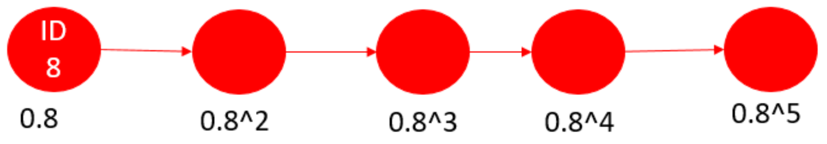


LinkID_length: As we are keeping track of every link(chain) in the graph, we are also storing the length of each linkID in a vector

Link Sensitivity Index: If we disturb one node (e.g., node-link id 8), how many nodes get affected. Which is nothing but the summation of all the terms given below. (sum of n terms in GP)



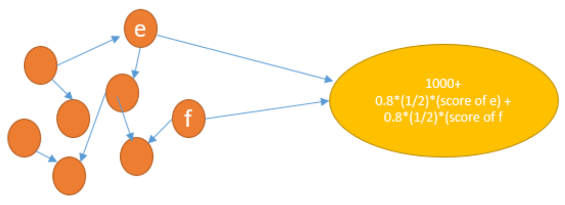
Assumption: Link connection is always linear



$$a + ar + ar^2 + ar^3 + \dots \qquad S_n = \frac{a(1-r^n)}{1-r}$$

Algorithm:

1. **Input the newly popped out node(x)**
2. **Input the in-degree of (x)**
    a. **Update the outdegree of source nodes that are responsible for indegree of (x)**
    b. **Assign the LinkID to (x) based on the convention discussed above**
    c. **After assigning the linkID, then update the length of that linkID that has been assigned in the linkID_length vector.**
    d. **Calculate the link Sensitivity index of source nodes and add it to a score of (x)**
    e. **Add [(80% score of each source node)*(probability of choosing that edge)] score to (x)**



[80% is chosen because the damping factor of page rank formulation is 0.85, (1/2) is multiplied because nodes 'e', 'f' has 2 outgoing edges and the probability of choosing one and reaching the yellow node(new node (x) here) is (1/2).
Remember, in randomized algorithms, the random surfer visits each node at least the mentioned number of simulation times. Here we considered the number of simulations parameter to be 1000, which

means we are trying to mimic the behavior of random surfer in randomized approaches]

3. **Input the outdegree of newly popped out node(x):**
    a. **Update the outdegree of (x) in the outdegree vector**
    b. **Assign the linkID to target nodes**
    c. **Update the LinkID Length of LinkId holding by the (x)**
    d. **Calculate the Link Sensitivity Index of the target and add the score to the target node**
    e. **Add [(80% score of new node(x))*(probability of choosing that edge from all the outdegree edges of (x))] score to target nodes**
4. **Take out the probabilities of each node by doing = (score at each node)/(total sum of scores of all the nodes in the graph)**
5. **Output the probabilities as final page rank vector**

Space requirement: O(N) -> for vector to store outdegrees of each node

+ O(N) -> for vector to store LinkID of each node

+ O(N) -> for vector to store length of each node

+ O(N) -> to store the result vector, where we update scores(visits) of each node.

N is the number of nodes in the graph at any given point in time.

Time complexity:

For every newly popped out node, we are doing O(1) work for each incoming link to the freshly popped out node and O(1) work for each outline that the newly popped out node has.

```
 For loop over indegree{
          O(1) time
}
+
For loop over outdegree{
          O(1) time
}
```

So, the time complexity is purely dependent on the in-degree or outdegree of each node.

Complete Graph:

If there are, n nodes in the graph and every time a new node pops out has all the n edges, and then for that node, the time complexity is going to be O (1 + 1+ 1 + ….) = O (n)

If from the 0th node the above situation continues, then the time complexity will be like this

O (1 + 2 + 3 + 4 + 5 + 6 + …..+ n) = O (n^2) at the end of whole graph.

* O (N) for each new node in worst case *

* O (N^2) for the whole graph*

N is the number of nodes in the graph at any given point in time.

Preferential Attachment model:

Considering Barabasi Albert model 'm' is taken to be constant, then every node takes O(1) time. Computaition of page rank for the whole graph will be O(n)

Let us look at finding correlation between two rank vectors using spearman coefficient;

Spearman Coefficien:

Formula

$$\rho = 1 - \frac{6\sum d_i^2}{n(n^2-1)}$$

ρ = Spearman's rank correlation coefficient

d_i = difference between the two ranks of each observation

n = number of observations

Note:

A **Spearman correlation** of **1** results **when** the two variables being compared **are** monotonically related, even **if** their relationship is not linear. This **means** that all data points with greater x values than that of a given data point **will** have greater y values as well.

Let us look at finding correlation between two rank vectors using spearman coefficient;
Spearman Coefficien:

Note:
A **Spearman correlation** of **1** results **when** the two variables being compared **are** monotonically related, even **if** their relationship is not linear. This **means** that all data points with greater x values than that of a given data point **will** have greater y values as well.

Data Set and calculations:

Please find pictures in excel sheets in this order:

1. Page rank calculated by gephi inbuilt function
2. Page rank by monte carlo algorithm
3. Dynamic approach

50nodes: https://drive.google.com/file/d/1Q8eQgozrlOyPCstrcG5VAKN3vR_1XHVO/view?usp=sharing

100 nodes: https://drive.google.com/file/d/1XFBeNE6oxttEO7ASgJ-i5FcFjcoVR98I/view?usp=sharing

500 nodes: https://drive.google.com/file/d/1Z-r4YxkWyUKm9JaNdc2mckKa_HHtTxST/view?usp=sharing

1000 nodes: https://drive.google.com/file/d/1g3X2W1-XUgVZ-X9xLVXDgN6NHAW-H903/view?usp=sharing

All the data sets have spearman correlation (r_s) = 0.999 to 1