Sequential Page Rank Algorithm

Indiana University, Bloomington

Fall-2012

A project report submitted to Indiana University

By

Shubhada Karavinkoppa and Jayesh Kawli

Under supervision of Prof. Judy Qiu

Table of Contents

1. Abstract

PageRank algorithm is used by Google search engine to measure the relative importance of the web pages on the web. It assigns a numerical weighting to each of the pages in the set. This algorithm is named after Larry Page. The PageRank of a page depends on the number pages that link to it i.e. every inbound link to a page increases its PageRank value.

2. Introduction

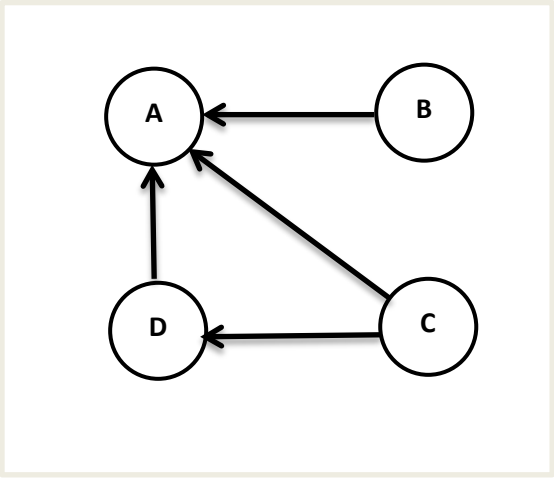
In this project we have implemented Sequential PageRank algorithm. The algorithm calculates the PageRank value of each page depending on the inbound links to that page. Each inbound link is like a vote. These votes are used to determine which pages are more important.
In this algorithm, calculations are done iteratively until the values converge. But in out implementation we have two options: until the values converge or until the specified number iterations are executed.

3. Problem Statement

We are given with an input file that contains 1000 pages along with the list of pages to which they are linked. This is similar to adjacency matrix. The aim of this project is calculate PageRank value of each page by taking into account dangling nodes and determine the ten most popular pages.

4. Theory

PageRank calculation is based on the graph of web where each webpage is like a node and each hyperlink is like an edge. Consider a web graph of 4 nodes A, B, C and D as shown in below figure. Initial PageRank values are assigned using probability distribution between 0 and 1 i.e. 1 divided by total number of web pages. In the following example, the initial PageRank for each page is 0.25.



Suppose the nodes B and D have links to A and C has links to A and D as shown in figure 1.1, then the PageRank of A is calculated as,

$$PR(A) = PR(B)/1 + PR(C)/2 + PR(D)/1$$

In general, the PageRank of any page u is given as,

$$PR(u) = \sum_{v \in Set} \frac{PR(v)}{L(v)}$$

The PageRank of u is dependent on the PageRank value of each page v in the Set (Set containing all nodes that have outbound link to u) divided by the number outbound links from page v.

The Damping Factor is taken into account when we consider that an imaginary surfer who is randomly clicking on the links will eventually stop. The probability that the person will continue is a damping factor and it is generally set to 0.85.

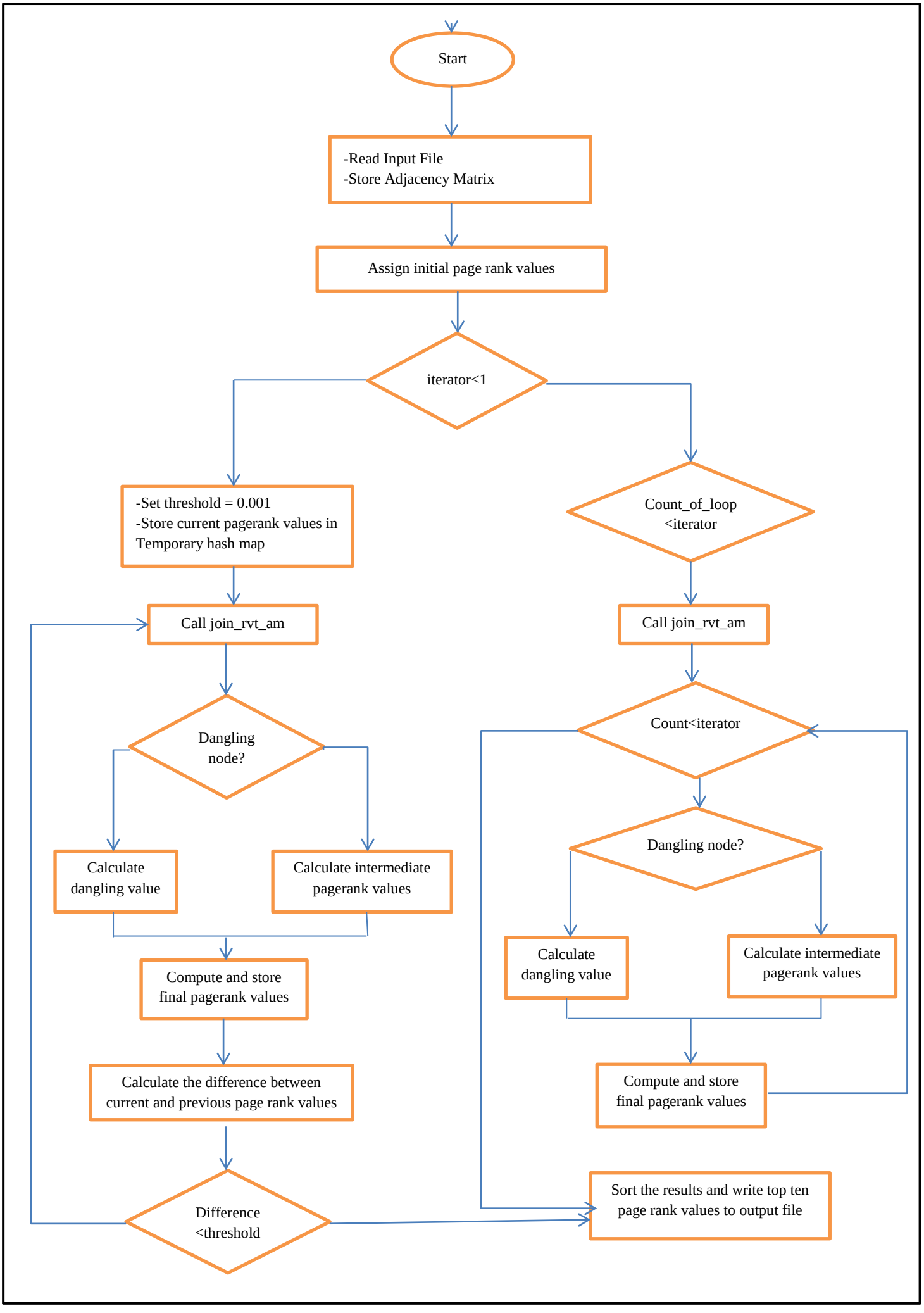
The PageRank formula considering damping factor is given as,

$$PR(u) = \frac{1-d}{N} + d * \sum_{v \in Set} \frac{PR(v)}{L(v)}$$

Dangling Links: Dangling link is simply a link to a page that does not have any outgoing links i.e. it does not link to any other page. These dangling links do not affect the PageRank value of any other page directly. Such links are kept aside from the system until PageRank values of all other pages are calculated. After all the PageRank values are calculated, they can be added back in without affecting significantly. The PageRank of all such links are added and then divided by the total number of webpages and this result is equally distributed among all the webpages to minimize the effect of dangling links.

5. Architecture

The below diagram shows the flowchart of program.


Start
-Read Input File
-Store Adjacency Matrix
Assign initial page rank values
iterator<1
-Set threshold = 0.001
-Store current pagerank values in Temporary hash map
Call join_rvt_am
Dangling node?
Calculate dangling value
Calculate intermediate pagerank values
Compute and store final pagerank values
Calculate the difference between current and previous page rank values
Difference <threshold
Count_of_loop <iterator
Call join_rvt_am
Count<iterator
Dangling node?
Calculate dangling value
Calculate intermediate pagerank values
Compute and store final pagerank values
Sort the results and write top ten page rank values to output file

6. Implementation

6.1 Reading command line input

We run page rank algorithm by giving string of input parameters though command line interface. These parameters are briefly described below.

args[0] = Name of the input file containing list of URLs and their association
args[1] = Name of the input file which stores Top 10 URLs along with their page rank value, total number of iterations and sum of all page rank values.
args[2] = Specifies total number of iterations to be done. However in implementation, this is not a strict requirement. Algorithm will automatically stop when convergence occurs.
args[3] = Damping factor – It represents the probability that random user will continue clicking on links is set to 0.85. Higher value will result in longer runtime while lower values results in short runtime.

Note: We have added functionality in this algorithm to improve its runtime efficiency. When user gives number of iteration as any nonzero positive value, it will calculate page ranks for specified number of iteration. However, this is not a strict requirement. When value of this parameter is less than 1, page rank calculations will continue till convergence.

6.2 Read the input file of given URLs

A name of the input file is given from command line input. This is the input file containing N URLs. For each node it is provided with the list of nodes to which is connected through outbound links. A node may or may not have an outbound link. In case node doesn't have an outbound value, we treat that node as dangling node and give special treatment in page rank calculation.
We use java functions FileInputStream, DataInputStream to create object of input file. Once object is created, file is read until null pointer is encountered.

6.3 Storing Graph of web pages in a Linked Hash Map

Once file is successfully read, we store all the nodes along with its neighbours in a LinkedHashMap data structure. Each node is stores as key and its associated successors are stored in a Arraylist data structure which acts as a LinkedHashMap value. One advantage of using LinkeHashMap is that, it stores the value in an order in which it has read from input file.

6.4 Number of URLs in input file

We count the number of URLs in the given input file by querying on size of LinkedHashMap which stores all the input URLs as a key. This value allows us to deal with dangling node issue and also results in close value of page rank for each web page.

6.5 Initializing Page rank values table

At the very first iteration of loop, page rank values of web pages are not known. As a part of initialization we set page initial page rank value as 1/N for all the input URLs. Where N is a total number of URLs.

6.6 Calculating actual page rank value

We will call the function
join_rvt_am() with following parameters

rank_values_table – Page rank table which store final values of page rank
adjacency_matrix_list – LinkedHashMap to store URLs along with their successors in the ArrayList data
url_size –Total number of input URLs
damping_factor – Usually set to 0.85

This function will calculate page rank value for each node in successive iteration using following formula

$$PR(u) = \frac{1-d}{N} + d * \sum_{v \in Set} \frac{PR(v)}{L(v)}$$

We first calculate intermediate page rank value for each URL and distribute average value of dangling URLs to each web page. We multiply this summation by damping factor d to reduce the pagerank of predecessor. In short term, when speaking of overall effect, a page loses its page rank by some extend when it has an outbound link to another page.

Once this is done, we add an additional term which corresponds to the probability that random user will click a page only given the total number of input links.

6.7 Number of Iterations

We give the input of how many iterations we require to do to get final and most clean value of page ranks for each page. However, this is not a strict requirement. We have introduced some changes to make the code more efficient so that it doesn't have to do ALL the input iterations even though convergence occurs beforehand.
Small value of iteration may give wrong results while large value might give more correct results but with more computation time.

6.8 Writing results to output file

After convergence occurred (Or loop has executed specific number of times) we write the results of computation in output file which includes,

A. List of Top 10 URLs along with their page rank values
B. Sum of all page rank values (Ideally equal to one)
C. Number of iterations occurred.

6.9 Optimizations and Efficiency boosting

Break the loop is solution converges
Instead of going over for loop for number of iterations given from command line, we try to limit the iterations until convergence happens.
Each time page rank values are calculated, we compare them against earlier page rank computations. If this difference goes below specific threshold (Set in the program) we break the loop and return results of iteration in output file.

6.10 Timing details

To get more insight regarding runtime behaviour of program, we introduced frequent checkpoints to get estimate of time required to calculate that function. We use this information to make our program more efficient in terms of time and space.

7. Discussion:

7.1 Result

List of Top 10 URLs was accumulated in output file along with their page rank value
As follows

Top 10 URLs with Highest Page Rank values

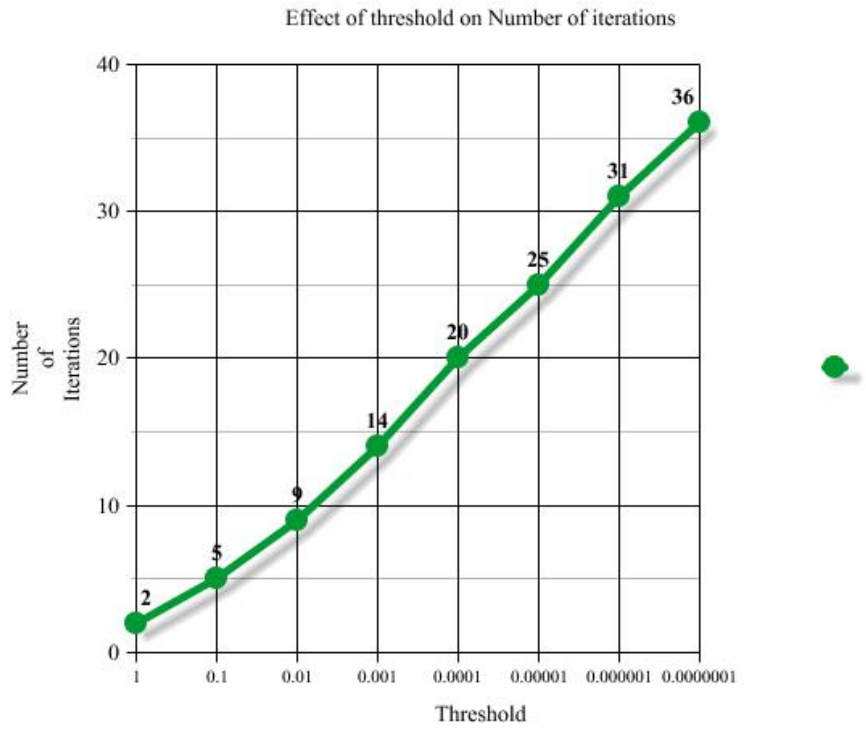
URL	Page Rank
4	0.13821304217473024
34	0.12302491704773691
0	0.11257935294330157
20	0.07736590523118934
146	0.05713176348278271
2	0.04792631126705502
12	0.02006643690709921
14	0.01790592635583653
16	0.01302811362009985
6	0.01295544157190792

Sum of individual Page rank values = 0.9999999999999800

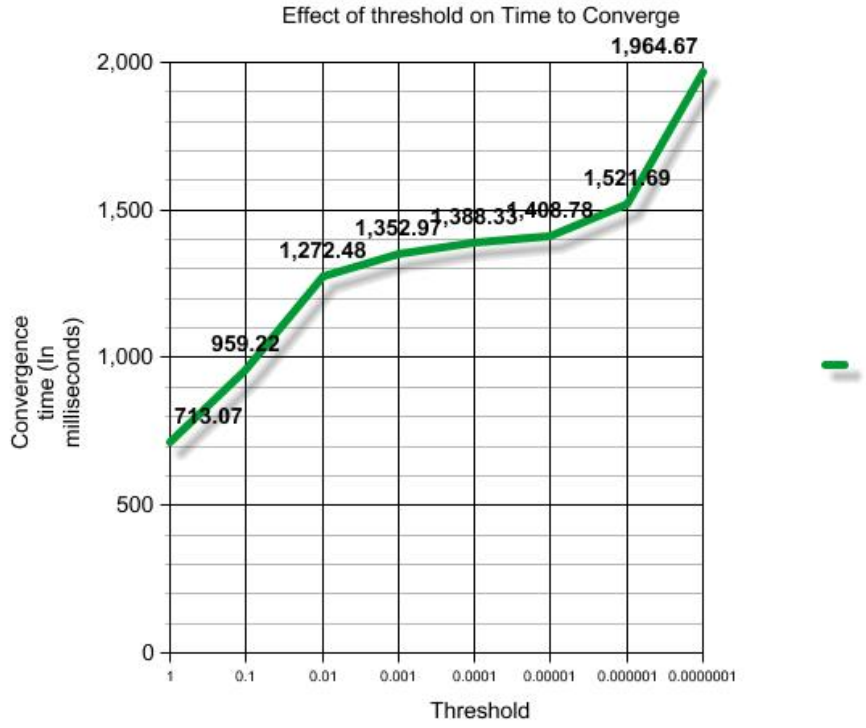
7.2 Performance analysis

Program converged for value well below given number of input iterations. Page rank values were calculated for only 14 numbers of times until convergence occurred for threshold value of 0.001

1. Following graph shows Number of iterations time as a function of threshold



2. Following graph shows Time taken to converge (In milliseconds) as a function of threshold value

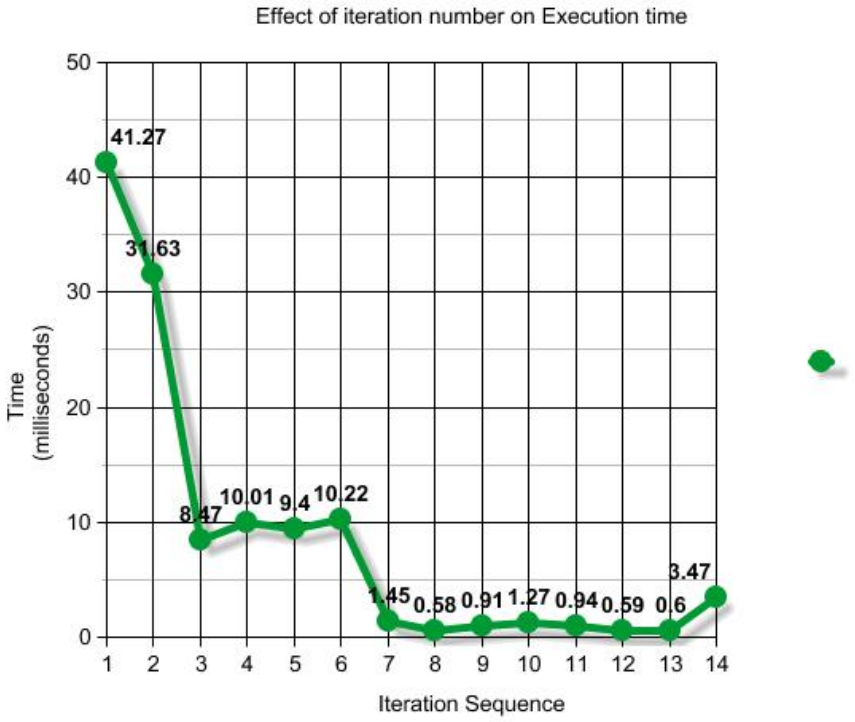


3. We ran this algorithm on input file of 1000 URLs (N=1000). Program ran successfully giving satisfactory set of results. sum of all page rank values evaluated to close to 1

Sum of all page rank values for threshold 0.001 = 0.9999999999999800

4. For threshold value of 0.001 and input iterations 50, program converged only in 14 iterations. Saving time on subsequent redundant calculations.

5. For threshold value of 0.001, page rank algorithm converged in 14 iterations. Following graph shows time taken (in milliseconds) as a function of iteration sequence.



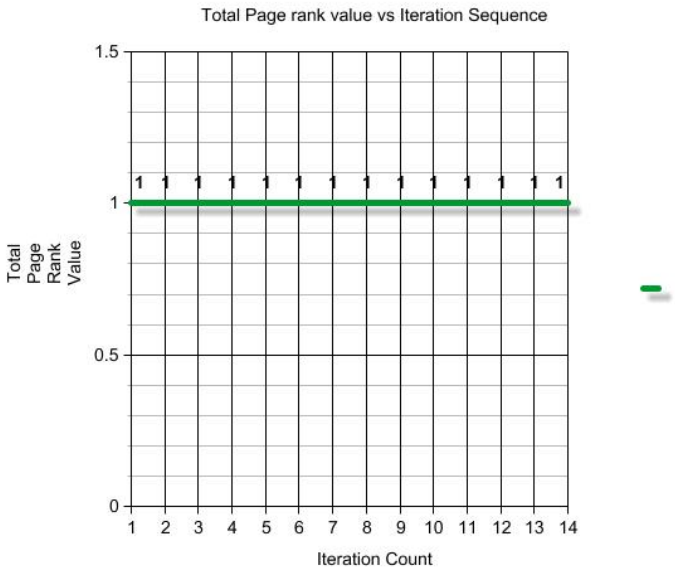
7.3 Findings

Thus we successfully implemented page rank algorithm which gives the top 10 URLs in terms of the page rank from given list of N input URLs. Following put forward our findings as follows

A. Program considers both regular nodes and dangling nodes with no outbound link
B. When page rank values of dangling nodes are ignored, we were still able to get expected list of top 10 URLs. However sum of observed page rank values thus obtained was much below 1 (0.75)
C. When we considered dangling nodes into our calculation, sum of 0.9999999999999800 was obtained which is very close to ideal value (1).
D. It was observed that, it is not at all necessary to run page rank algorithm for any number of times because in some cases if node configuration is ideal, convergence occurs even before specified number of iterations.
E. For fixed values of threshold and iteration count of 50, convergence occurred only at 14^{th} iteration.

F. We get good approximation of page rank algorithm after algorithm is ran for fixed number of times when convergence happens
G. Minimum value of page rank could be (d-1)/N according to given formula when it has no inbound links
H. Maximum value of page rank can be ((d-1)/N)+(d*(N-1)) when all remaining (N-1) nodes point only to current node
I. If the page has more number of inbound links, then it results in increasing page rank of that page which is evident from given page rank calculation formula

We calculated cumulative page rank value going through each iteration. It was observed that, this value always remain close to one



Graph of cumulative page rank value versus sequence of iterations

8. Assumptions:

To maintain fairness in page rank calculation, we introduce following assumptions in algorithm implementation

A. For the given pair of web pages, one page cannot have more than one outbound links on the same page
B. Likewise, one page cannot have more than one inbound links from same page
C. Any web page cannot have outbound link pointing to itself

Assumptions mentioned above are necessary, as it is possible for any website administrator to exploit its links to elevate overall page rank in unfair manner.

9. Conclusion and Future Work

A. We implemented page rank in sequential manner. However, performance and efficiency of this implementation decreases as we increase the number of input URLs in given graph. As a part of future work, we can implement this algorithm in parallel fashion which is much more time efficient than its sequential counterpart.

B. These algorithms when tested on file containing N=1000 URLs, gave excellent results in terms of final list of top 10 webpages. However, behaviour of this project is still unknown when relatively large input file (N>100000) is given. We will try to study its behaviour on large inputs and introduce modifications if it does not comply with standard performance.

10. Acknowledgements

Completing any project is not possible without support and help from many people. Page rank algorithm is no exception to this rule.

We would like to thank Prof. Judy Qiu, Assistant Professor of Computer Science and Informatics at School of Informatics and Computing at Indiana University for helping us to develop an idea of this wonderful algorithm. Also, presentation slides thus provided on course web page gave much more insight about functionality of this algorithm.

We also appreciate the help from Ila Jogaikar, an Associate Instructor for B534-Distributed System for her patience and co-operation throughout this project.

Finally we would like to thank all those anonymous people all around the internet, who shared their knowledge and experience on various blogs and webpages. This helped us to know background and current implementation of the page rank algorithm.

11. References

http://en.wikipedia.org/wiki/PageRank
http://www.webworkshop.net/pagerank.html
http://www.sirgroane.net/google-page-rank/
http://pr.efactory.de/e-pagerank-algorithm.shtml
http://infolab.stanford.edu/~backrub/google.html