PageRank Algorithm for Dynamic Graph

Priyanka Tayade
W1093927
Directed Research, COEN 493
Instructor: Dr. Nam Ling

Department of Computer Science and Engineering
Santa Clara University
ptayade@scu.edu

The initial PageRank algorithm was proposed by Sergey Brin and Lawrence (Larry) Page (co-founder of google) in their Stanford university paper called "The Anatomy of a Large Scale Hypertextual Web Search Engine" [1] since then there have been a lot of modification to the initial algorithm. During that time the web content was mostly static. But the scenario has changed today, a large amount of web data is constantly changing in content as well as in structure. The search engines need to keep PageRank up-to-date. Google updates its PageRank roughly every month famously knows as Google Dance. In this paper we will discuss approach for estimating PageRank value when the graph is changing.

BACKGROUND:

Google computes its search index based on 200 parameters one of which is PageRank. Google also index the pages based on the quality of content. The concept of PageRank is that the pages are connected on the internet though hypertext links pointing to one another. This forms a graph of internet where pages are considered as nodes and edges are considered as links. PageRank utilizes the structure of web for ranking the pages. If a page is pointing to another page then it is considered to cast a vote for the page it is pointing to. If a Page is pointing to more than one page then the vote is equally divided among them. Pages with more number of backlinks are considered more important. But this is not always true, page with less number of backlinks can still have higher PageRank than the page with more number of backlinks if the PageRank of the backlinks to the first page is more. PageRank is computed using a random surfer model. Imagine a surfer surfing the internet so the probability that the surfer will click on a particular link is random given by $1/O_n$ where O_n is the number of outlines from that page. The surfer is assume to eventually get bored of visiting the same page again and again, and would stop clicking is given by the damping factor d, this solves the problem of cycles in the graph. PageRank of a page can be computed using the following formula.

$$PR(p) = (1-d) + d\ (PR(Ln)/O(Ln) + ... + PR(Ln)/O(Ln))) \quad [2]$$

Where
PR(p) : PageRank of a page p
R(Ti) :PageRank of pages Ti which link to page P.
O(Ti): outbound links of page Ti
d: Damping factor {0 < d < 1}

This formula was presented in the first paper by Sergey Brin and Larry Page which was later corrected to PR(p) = (1-d) /N + d (PR(Ln)/O(Ln) + ... + PR(Ln)/O(Ln))) where N is the total number of nodes on the network. (1-d) /N shows the probability that the random surfer will navigate to this page. In the first formula the sum of the PageRank values add up to N and in the second formula it adds up to 1. Sergey Brin and Larry and other members at google claimed that the sum of all PageRank is one hence supports the second formula.
The above formula also can be represented in matrix form. Let M be the conductance matrix for a web graph G. M is NxN matrix where N is the total number of nodes (pages) in the graph.

$Pr = d.M.Pr + (1-d)/N$

The conductance matrix represents the web structure of the graph. The value for each outgoing edges is 1/n where n is the number of outing edges. Values in the matrix element Mij is the value of an outgoing edge from node i to j. The sum of all the rows for a particular column will be equal to 1 which is equal to the initial value of each node. (1-d) represents the teleport probability and 1/N represents the initial probability.

PROBLEM:

About 50% of web content and 80% of the web structure is added new yearly (*The Evolution of the Web from a Search Engine Perspective).* Due to this reason it is very essential that search engine has its PageRank up to date for producing quality results. Although computing PageRank is a tedious process. This is done by distributed servers and can take up to days for the algorithm to converge on the real PageRank value. This is done periodically by google. Approximately the size of the graph can be 10^{10} pages. The google crawler crawls though the internet through the list of URL already recognized for change in content and new URLs. If the new URLs are found then they are added in the list and crawled later. We will limit our discussion on PageRank particularly on the problem of estimating the real PageRank value if the graph is changing.

PROBING

Bahman Bahmani, Ravi Kumar, Mohammad Mahdian and Eli Upfal in their paper [3] on evolving graph suggests an algorithm to determine PageRank in the evolving graph by probing. They described four different approaches for probing two of which are deterministic and other two are non-deterministic. Random probing would be the baseline for proportionally probing algorithm for analysis and both are non-deterministic algorithm. Round robin probing is the baseline for priority probing algorithm and these two are deterministic.

In Random probing if we start from a node at random then the next edge the algorithm will traverse will be totally random in nature. In random probing we probe nodes with equal frequency, we can improve our base case in proportional probing by randomly probing the outgoing edges of the page with higher rank nodes with higher frequency. Changes in the outgoing edges of the PageRank with higher frequency effects the PageRank of other pages more than those with lower PageRank. In round robin probing we cycle through each node and probe it in order with equal frequency. Improvement to this deterministic approach is nodes with higher priority being probed first. After a node is probed its priority is set to zero. There is another hybrid algorithm which is combination

of proportional probing and round robin probing. The value of beta is used to set the ratio of proportional probing to round robin probing, value of beta ranges between, 0 and 1.

Experiments shows that the proportional and priority probing works far better than random and round robin probing respectively. Since the changes are happening in random on the graph it was expected that random or round robin probing should work at advantage. But in reality proportional and priority seems to perform better this shows that real network are biased to high PageRank nodes.

The performance of hybrid algorithm as far better than both proportional and round robin algorithm. The best performance is when beta is set to 0.9. By this method the nodes with higher PageRank is probed frequently and the round robin approach makes sure that each node is probed at least once. The change in frequency of probing and the frequency of change in graph does not seem to effect performance of any of the above discussed algorithms differently. But any of the algorithm works.

DECENTALIZED PAGE RANKING (DYNA-RANK)

The traditional approach for computing PageRank is centralized which requires a huge amount of time may be days for computing the PageRank because the entire internet is considered as a graph. Mahendra Kale and Mrs. P. Santhi Thilagam talks about decentralized approach for updating PageRank in their paper [4], "DYNA-RANK: Efficient calculation and updation of PageRank". The block rank shows that the web has the nested hyperlink structure [4]. About 79.1% of the hyperlink are intra-host which means they are connect to each other within same hub. And about 20.9% are inter-host which means they are connect to page from different host. It is also found that even large number (83.9%) of links are intra-domain links.

Experiments shows that computing DYNA-RANK on distributed peers takes less time than computing PageRank on the whole centralized graph. This is because many of the links are intra hub or domain so many times the changes due to addition/deletion of links or pages make not have the same effect on the PageRank of other domain or hub. Therefore calculating the PageRank of Stanford University site due to some of the changes in Mumbai University site is not required as the effect of change in PageRank suffers from propagation decay.

A set of domain or a set of hubs can be considered as one peer and this way there can be multiple peers. At first we will not consider links which are inter peer. We will label each peer, peers will have total number of nodes (pages) given by $n_{(\text{peer i}).}$ Initially the rank of each page is given as 1. After calculating DYNA-RANK for each peer independently by power method. We will now connect all the peers with the help of inter peer links and the overall Rank of the peer graph is calculated by the following formula

$$New\ Rank\ (K, L) = \frac{PR\ (K)}{(n\ (K)\ peer(i)) + 1}$$

New Rank (K , L) is the new rank from node K to L. node K is in peer i and node L is in peer j and i ≠ j. *n* (*K*)*peer*(*i*)) is the number of out links from node K. PR(K) is the PageRank of node k.

If there is any change in one peer then if the relative change is larger than the tolerance value of the ε then we need to update the changes to other peer otherwise the changes which are less than ε do not affect the rank of other peer.

$$RC = \frac{abs\ (new\ rank - old\ rank)}{(New\ rank)}$$

If RC > ε update the changes. If the value of ε is kept large then the number of iterations required are less hence PageRank is computed faster. But smaller the value of epsilon the closer is the value of PageRank to its real value.

UPDATED PAGERANK BY ITERATIVE AGGREGATION

Amy N. Langville and Carl D. Meyer discovered a new method for updating PageRank by iterative method [5]. They were able to update PageRank by 25% to 14% of the time which was required by googles PageRank algorithm.

Web graph represents Markov chain [6] which can be represented by Markov transition matrix. A Markov chain is a stochastic process with the Markov property (Wikipedia). In Markov chain a process transition from one state to another and the probability of the next state depends solely on the current state and not on any preceding chain of events this property of "memoryless-ness" is known as Markov property and the PageRank problem is one of the example of the Markov chain.

$$Q_{\mathbf{mxm}},\ \boldsymbol{\phi}^T = (\phi_1, \phi_2, \phi_3 \ldots \phi_m) \text{ where time is t}$$

$$P_{\mathbf{nxn}},\ \boldsymbol{л}^T = (л_1, л_2, л_3 \ldots л_n) \text{ where time is t +1}$$

Where m ≠ n, as number of nodes can be different in both the matrices due to addition or deletion of pages. Q and P, are the graph structure which is known at time t, also $\boldsymbol{\phi}^T$ the initial PageRank is known. We need to calculate $\boldsymbol{л}^T$ updated PageRank which is not known.

This algorithm states that there exists a partition S such that $S = G \cup \bar{G}$ for which the number of iterations the algorithm takes to updates the PageRank is strictly less than the number of iterations required by Googles power method. We partition the Markov chain in states G and $\bar{G}$ at time t+1 such that P is square irreducible stochastic matrix partitioned as follows.

$$P_{nxn} = \begin{matrix} G \\ \bar{G} \end{matrix} \begin{pmatrix} P_{11} & P_{12} \\ P_{21} & P_{22} \end{pmatrix}$$

To understand this please refer to theorem 10.1 in paper stochastic complementation, uncoupling Markov chains, and the theory of nearly reducible systems by C. D. Mayer.

G has all the newly added states and some old states, g states un-aggregated and $\bar{G}$ has n-g states aggregated. For chain C, the stationary distribution problem we need to determine the unique vector $\pi_1 \times n$ in this case PageRank vector which satisfies

$$\pi P = \pi,\ \pi i > 0,\ \sum_{i=1}^{n} \pi i$$

If the chain is smaller this problem can be easily solved using linear equation. For chains such as the web size graph where states can be replaced by pages the standard method will be tedious. For such large scale problem we uncouple the original m states in to smaller chains say C_1, $C_2 \ldots$ C_k which has $\pi_1, \pi_2 \ldots \pi_k$ states and $\sum_{i=1}^{k} \pi_i = m$. These uncoupled chains will have properties as follows.

- Smaller chains C_i should be irreducible and has unique distribution vector given by S_i.

  $$S_2^T = \frac{(\Phi_{g+1}, \ldots \Phi_n)}{\sum_{i=1}^{k} \Phi_i}$$

  Aggregated transition matrix A of size (g+1) x (g+1) is given by

  $$P_{nxn} = \begin{matrix} G \\ \bar{G} \end{matrix} \begin{pmatrix} P_{11} & P_{12}e \\ S_2^T P_{21} & 1 - S_2^T P_{22} e \end{pmatrix}$$

  Where e is the vector of all ones

- S_i is completely independent of each other where i in this case ranges from 1 to g and n-g to n respectively, it can be calculated in parallel.
- Stationary distribution vector π for the larger chain which is the PageRank itself can be easily produced by coupling smaller stationery distribution vectors S_i

According to Simon and Ando in Aggregation of variables in dynamic systems gives following very important analysis.

- Short-run Evolution- the components of final stationary distribution vector are made of smaller distributions Si having closely coupled chains independently tend toward their own local equilibrium. The local equilibrium is proportional to values in Si.
- Short-run stabilization interval I: During this interval I the chain is approximately stable. $\pi n = (\alpha 1 s1 \quad \alpha 2 s2 \cdots \quad \alpha k sk)$ where α are constant which depend on previous $\pi 0$
- Middle-run evolution: As n > I πn moves away from short run stabilization vector given in previous equation and but it maintains approximate relative stability between components of closely coupled chains which is given by. $\pi n = (\beta\ 1 s1 \quad \beta\ 2 s2 \cdots \quad \beta\ ksk)$
- Long run evolution: The values of β settles down which defines the stationary vector for the complete chain.

Successful implementation of these ideas relies on our choice of partition S. experiments shows that for a good partition this approach works very well. For California.dat the data for search query California which has 9664 nodes and 16150 links the updated PageRank algorithm taking |G| = 1500, works in takes 14 iterations for time 1.42 whereas Googles PageRank takes

176 iterations and time taken is 9.63. If we choose some bad value of |G| = 5 for abortion.dat algorithm took as much time as taken by Googles PageRank. Good value of G for abortion.dat was supposed to be 50. For detail please refer to paper by [5]. Choosing a good value of G is the active area of research.

IMPROVING EFFICIENCY

PageRank is calculated by either power method (or iterative method) of by algebraic method. Even though the power method is slow due to the number of iterations by algebraic method it is difficult to compute with matrix of size of internet graph. There have been several research on to improve computation time by reducing the number of iterations or by reducing the computation required in each iteration. It is seen that it is difficult to achieve both as one is a tradeoff of other. If we reduce number of iteration we end up increasing the computation time and vice versa. There has been several studies on this subject and a group of researchers at Stanford were able to reduce the number of iteration. Kumar and et all' found that for most of the pages, PageRank converges to their true value very early and only because of some pages which takes a longer time to converge the algorithm keeps on performing further computations till each page reaches to its true PageRank value. The improved method would be if the PageRank reaches its convergence criterion then the PageRank of that page is locks and the computation is skipped. Hence we will only have to compute for the remaining pages who are not reached to their convergence criteria yet. This method was successful to reduce computations by 17%.

There is a method to extrapolate the PageRank, it helps reduce computation time by 50—300%. This method works by observing the power iterations by spectral decomposition googles. Spectral decomposition means that each component of the PageRank vector will be separated and we observe that we have the real value of PageRank much in the beginning but it is only dude to other components of the equation which takes time to cancel out after the iterations. If we could guess the values of these components then we can avoid the cost of these iterations and get to the real PageRank value much faster by cancelling the components by ourselves.

$$\boldsymbol{\pi}^{(k)T} = \boldsymbol{\pi}^{T} + \lambda^{k}{}_{2}\gamma_{2}\mathbf{y}^{T}{}_{2} + \lambda^{k}{}_{3}\gamma_{3}\mathbf{y}^{T}{}_{3} + \cdot\ \cdot\ \cdot + \lambda k n \gamma n \mathbf{y} T n,$$

From the above equation we can see that we already have $\boldsymbol{\pi}^{T}$ but due to $\lambda^{k}{}_{2}\gamma_{2}\mathbf{y}^{T}{}_{2}$ its value is not available. But theoretically it was found that we can guess the value of $\lambda^{k}{}_{2}\gamma_{2}\mathbf{y}^{T}{}_{2}$ by parameters which we already know. We need k+1 and k+2 i.e. 2 steps to predict the value from following equation.

$$\lambda^{k}{}_{2}\gamma_{2}\mathbf{y}^{T}{}_{2} \approx \frac{(\boldsymbol{\pi}(k+1)T - \boldsymbol{\pi}(k)T).2}{(\boldsymbol{\pi}(k+2)T - 2\boldsymbol{\pi}(k+1)T - \boldsymbol{\pi}(k)T)}$$

There is one more method introduced by the same group of researchers Kumar and et al' called the block methods along with improving the work per iteration and reducing the number of iterations of power method. This block method recognizes the web graph as lumped in hosts and domain and is likely the inspiration behind DYNA-RANK which we just studied. This method also known as aggregation method and these methods along with other numeric method to improve

efficiency are explained in depth in *Google's PageRank and Beyond: The Science of Search Engine Rankings.* [7]

EXPERIMENT: For better understanding of various notions discussed above let us run some test cases, following is the directed graph which we will use to run the test cases.

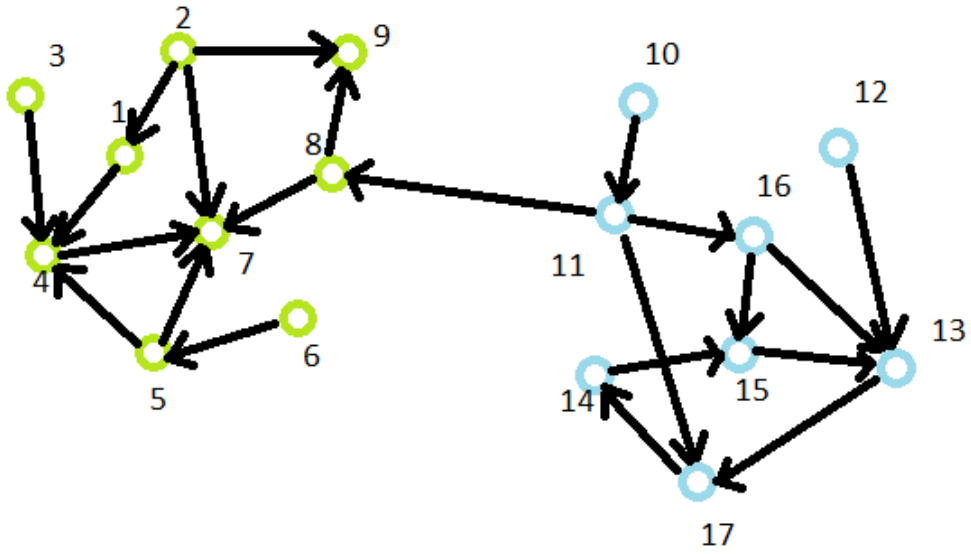

Figure 1: DGA for test

We can produce matrices M (17 x 17) from the above graph where nodes 1 to 17 will represent rows and columns.

Matrix M with dangling nodes representing graph G

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	0	0.333	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	1	0	1	0	0.5	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0.333	0	1	0.5	0	0	0.5	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0.333	0	0	0	0	0	0.5	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0.5	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
15	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0.5	0

16	0	0	0	0	0	0	0	0	0	0	0.5	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0.5	0	1	0	0	0	0

The above matrices contains dangling nodes. The value of the PageRank passed to these node is not propagated further in graph. In surfer model, if a surfer reach a page which does not link to any other page further he just simply jumps to some other page at random and again starts surfing. To solve this problem we can calculate the PageRank of other nodes and then add these nodes later. We can also imagines that these dangling nodes have edges to all the other nodes in the graph this way we can distribute their weight throughout the graph.

Matrix M without dangling nodes representing graph G

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	0	0.333	0	0	0	0	0.05882	0	0.05882	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0.05882	0	0.05882	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0.05882	0	0.05882	0	0	0	0	0	0	0	0
4	1	0	1	0	0.5	0	0.05882	0	0.05882	0	0	0	0	0	0	0	0
5	0	0	0	0	0	1	0.05882	0	0.05882	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0.05882	0	0.05882	0	0	0	0	0	0	0	0
7	0	0.333	0	1	0.5	0	0.05882	0.5	0.05882	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0.05882	0	0.05882	0	0	0	0	0	0	0	0
9	0	0.333	0	0	0	0	0.05882	0.5	0.05882	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0.05882	0	0.05882	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0.05882	0	0.05882	1	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0.05882	0	0.05882	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0.05882	0	0.05882	0	0	1	0	0	1	0.5	0
14	0	0	0	0	0	0	0.05882	0	0.05882	0	0	0	0	0	0	0	1
15	0	0	0	0	0	0	0.05882	0	0.05882	0	0	0	0	1	0	0.5	0
16	0	0	0	0	0	0	0.05882	0	0.05882	0	0.5	0	0	0	0	0	0
17	0	0	0	0	0	0	0.05882	0	0.05882	0	0.5	0	1	0	0	0	0

I have used PageRank MATLAB function from [8] http://en.wikipedia.org/wiki/PageRank which is shown as below.

```
function [v] = PageRank(M, d, v_quadratic_error)

N = size(M, 2);
v = rand(N, 1);
v = v ./ norm(v, 1);
last_v = ones(N, 1) * inf;
M_hat = (d .* M) + (((1 - d) / N) .* ones(N, N));

while(norm(v - last_v, 2) > v_quadratic_error)
    last_v = v;
```

```
    v = M_hat * v;
end
```

Test Case 1: We run the PageRank function in MTALAB and document the PageRank value for each node in the following table.

Input:

M: the matrices M which represents the graph shown in figure 1. We used the matrices which distributes the weight of dangling nodes throughout the graph.

v_quadratic_error= 0.001 (this is the convergence criteria, less the value more are the iterations required and more accurate is the value of PageRank)

d=0.85 (0.85 is considered as the best value of d)

We use this input throughout but with some changes which will be mentioned as we proceed.

Page Label	Incoming links	Outgoing links	Weights of incoming link	PageRank
1	2	1	0.45098	0.017933
2	1	3	0.117647	0.013973
3	1	1	0.117647	0.013973
4	3	1	2.617647	0.05208
5	2	2	1.117647	0.025851
6	1	1	0.117647	0.013973
7	5	17	2.45098	0.079126
8	2	2	0.117647	0.013973
9	3	17	0.95098	0.023871
10	1	1	0.117647	0.013973
11	2	2	1.117647	0.025851
12	1	1	0.117647	0.013973
13	4	1	2.617647	0.173627
14	2	1	1.117647	0.160116
15	3	1	1.617647	0.160573
16	2	2	0.617647	0.02496
17	3	1	1.617647	0.172173
Total				1

Table 1: Table documents our observation for test case 1.

OBSERVATION: In table 1 first column contains the numeric label for each node. Second column contains the number of links pointing to that node. Third column shows the number of nodes that particular node points out to. Fourth column shows the total weight of the links that are incoming. Last column shows the PageRank obtained by running the PageRank algorithm on matrices M. The total at the bottom of the table is the total of PageRank value of each node which sums up to 1 as stated by Larry Page and S. Brain in their first paper.

We observe that if more number of links are pointing to a node then the weight of that incoming links for that page/node is higher and also the PageRank value of that node is higher. We can see that 4, 7, 9, 13, 15, 17 has more than 3 incoming links therefore their PageRank value is higher.

We can also observe that even though 7 has highest incoming links i.e. 5 the PageRank of 13, 17 is higher than 7 which is only 4 and 3 incoming links respectively. This shows that the value of PageRank not only depends on the number of incoming links but also on the rank of the incoming links.

Test Case 2: We keep other inputs same but only change the value of d to 0, 0.5, 0.85, and 1.

	D=0	D=0.5	D=0.85	D=1
	0.0588	0.039473	0.017933	0.0016
	0.0588	0.033832	0.013973	0.0012
	0.0588	0.033832	0.013973	0.0012
	0.0588	0.083193	0.05208	0.0052
	0.0588	0.050754	0.025851	0.0024
	0.0588	0.033832	0.013973	0.0012
	0.0588	0.102253	0.079126	0.0086
	0.0588	0.033832	0.013973	0.0012
	0.0588	0.047933	0.023871	0.0022
	0.0588	0.033832	0.013973	0.0012
	0.0588	0.050754	0.025851	0.0024
	0.0588	0.033832	0.013973	0.0012
	0.0588	0.105835	0.173648	0.2425
	0.0588	0.083427	0.160093	0.2411
	0.0588	0.087109	0.160875	0.241
	0.0588	0.046527	0.02496	0.0024
	0.0588	0.099749	0.171873	0.2431
Total	1	1	1	1

Table 2: The table shows the value of page rank as we change the value of d

OBSERVATION:
Case D=0: We get the probability distribution of random surfer linking on particular link which is 1/17=0.0588 in this case.
Case D=1: The PageRank value becomes very sensitive to changes on the graph which is not tested in our experiment.
Case D=0.85: It has been proved that as we go higher than 0.5 we get better PageRank value and at 0.85 it is considered to be ideal choice of d. As we go higher than 0.85 PageRank becomes sensitive to changes.

Test Case 3: We have taken graph such that some parts of graph are more connected to each other than other parts. Nodes 1 to 10 are more connected and nodes 11 to 17 are more connected. There is one edge 11 to 8 which connect the networks together to form the graph above. Idea behind this is that real network will be somewhat similar to your test graph where some nodes have dense interconnection among each other. We want to see how these nodes are affected particularly which

are away from the nodes or edges which are updated. In this test case we will check the effect of addition and deletion of edges.
M: the matrices M (we will add or delete edges)
v_quadratic_error= 0.001
d=0.85

	1	2	3	4	5	6	7	8	9
	ORIGINAL	DELETE EDGE 16 TO 15	COLUMN 1-2	ADD EDGE FROM 16 TO 12	COLUMN 1-4	ADD EDGE 16 TO 11 (DELETE FROM 16 TO 12)	COLUMN 1-6	ADD EDGE 16 TO 11 AND 14 TO 11	COLUMN 1-8
1	0.017933	0.017933	0	0.017933	0	0.017933	0	0.017974	-4.1E-05
2	0.013973	0.013973	0	0.013973	0	0.013973	0	0.014002	-2.9E-05
3	0.013973	0.013973	0	0.013973	0	0.013973	0	0.014002	-2.9E-05
4	0.05208	0.05208	0	0.05208	0	0.05208	0	0.052244	-0.00016
5	0.025851	0.025851	0	0.025851	0	0.025851	0	0.025917	-6.6E-05
6	0.013973	0.013973	0	0.013973	0	0.013973	0	0.014002	-2.9E-05
7	0.079126	0.079126	0	0.079126	0	0.079126	0	0.079443	-0.00032
8	0.013973	0.013973	0	0.013973	0	0.013973	0	0.014002	-2.9E-05
9	0.023871	0.023871	0	0.023871	0	0.023871	0	0.023932	-6.1E-05
10	0.013973	0.013973	0	0.013973	0	0.013973	0	0.014002	-2.9E-05
11	0.025851	0.025851	0	0.025851	0	0.033891	-0.00804	0.109353	-0.0835
12	0.013973	0.013973	0	0.021046	-0.00707	0.013973	0	0.014002	-2.9E-05
13	0.173648	0.176387	-0.00274	0.172355	0.001293	0.167829	0.005819	0.125555	0.048093
14	0.160093	0.162528	-0.00243	0.159271	0.000822	0.158915	0.001178	0.156414	0.003679
15	0.160875	0.152769	0.008106	0.156241	0.004634	0.157732	0.003143	0.097438	0.063437
16	0.02496	0.02496	0	0.02496	0	0.028377	-0.00342	0.060317	-0.03536
17	0.171873	0.174805	-0.00293	0.171547	0.000326	0.170552	0.001321	0.1674	0.004473
18									
SUM	1	1		0.999998		0.999999		1	

Table 3: Table shows the effect on PageRank with addition and deletion of edges

OBSERVATIONS:

DELETE EDGE 16 TO 15: There are two edges from 16, we deleted one edge from 16 to 15 and another edge from 16 to 13 is kept as it is. All the PageRank is now passed to 13 and since 13, 14, 15, 17 are connecting in direction of edge from 16 we see that PageRank of 13, 14, 15 and 17 are change and all the other PageRank remains the same. Thus the effect of this change was local and did not propagate throughout the graph. Column 3 shows the change in PageRank value.

ADD EDGE FROM 16 TO 12: Similarly when an edge is added from 16 to 12 we can observe same local change and the rest of the PageRank remained unchanged. We can see that the PageRank of the connecting link 11 and 8 is not changed.

ADD EDGE 16 TO 11: Now we change the direction of edge from 11 to 16 to point from 16 to 11. We observe that this will change the PageRank of 11 but still this change was not sufficient enough to propagate the change through node 8 and rest of the graph PageRank remains unchanged.

ADD EDGE 16 TO 11 AND 14 TO 11: We now want to add more edge to point to 11 which will change the PageRank of 11 enough to change the PageRank of 8 and hence the PageRank of nodes which are closely connected to 8 are effected and now the PageRank of whole graph has changed which can be seen in last column.

Even though the changes shown in last column of the table is small this shows that if we monitor these critical nodes such as 8 and 11 for changes we can predict that if the overall PageRank of the graph is going to change or not. Since the changes seems to effect the connected nodes more than other nodes which are far and less connected to the nodes where change are occurring. By exploiting the structure of the graph and its connectivity we can predict and keep on updating the PageRank as an when the changes happen. Applying this to web size graph is more complex and needs to be studied in much detail.

Test Case 4: Now we want to explore the addition of a node and its effect on the PageRank.

	1	2	3	4	5	6	
	ORIGINAL	NEW NODE 18 POINTING TO 11	COLUMN 1-2	ADD EDGE 16 POINTING TO 18	COLUMN 1-4	DELETE EDGE FROM 16 TO 15	COLUMN 1-6
1	0.017933	0.016404	0.001529	0.016405	0.001528	0.016404	0.001528
2	0.013973	0.012783	0.00119	0.012783	0.00119	0.012783	0.00119
3	0.013973	0.012783	0.00119	0.012783	0.00119	0.012783	0.00119
4	0.05208	0.047642	0.004438	0.047643	0.004437	0.047642	0.004437
5	0.025851	0.023648	0.002203	0.023648	0.002203	0.023648	0.002203
6	0.013973	0.012783	0.00119	0.012783	0.00119	0.012783	0.00119
7	0.079126	0.072383	0.006743	0.072384	0.006742	0.072383	0.006742
8	0.013973	0.012783	0.00119	0.012783	0.00119	0.012783	0.00119
9	0.023871	0.021837	0.002034	0.021837	0.002034	0.021837	0.002034
10	0.013973	0.012783	0.00119	0.012783	0.00119	0.012783	0.00119
11	0.025851	0.034513	-0.00866	0.041879	-0.01603	0.046229	-0.01603
12	0.013973	0.012783	0.00119	0.012783	0.00119	0.012783	0.00119

13	0.173648	0.1718	0.001848	0.164578	0.00907	0.162388	0.00907
14	0.160093	0.160156	-6.3E-05	0.157317	0.002776	0.156957	0.002776
15	0.160875	0.161193	-0.00032	0.155681	0.005194	0.146666	0.005194
16	0.02496	0.027451	-0.00249	0.030581	-0.00562	0.03243	-0.00562
17	0.171873	0.173495	-0.00162	0.169903	0.00197	0.170154	0.00197
18		0.012783	-0.01278	0.021448	-0.02145	0.026566	-0.02145
SUM	1	1		1		1	

Table 4: The table shows the effect on PageRank with addition and deletion of nodes.

OBSERVATIONS:

NEW NODE 18 POINTING TO 11: By the addition of node 18 and making it point to 11 we change the PageRank of overall graph. This happens by two reasons, one due to the change in number of incoming nodes to 11 which will increase its PageRank and second is, if we recall the formula of PageRank we can see that it also depends on total number of nodes in the graph which increases to 18 in this case. Therefore the PageRank of all the nodes are bound to change.

ADD EDGE 16 POINTING TO 18: Now we add another edge form 16 to 18 this will again change the PageRank

DELETE EDGE FROM 16 TO 15: Now again as we deleted edge from 16 to 15 we can observe that the overall PageRank was changed by addition of new nodes in value but it does not change relatively.

SUMMARY:

As the volume of data keeps on increasing and changing constantly this terabytes of data which is needed to compute the Page Ranking for a good search results will challenge us more to further improve our ranking schemes. There are several other ranking schemes which are not discussed in this paper such as HITS which is based on hubs and authority model but are worth exploring. In this paper we studied Googles PageRank algorithm and its background. We also looked at better crawling scheme for updating PageRank by polling and we also saw decentralized form of PageRank or BlockRank by studying DYNA-RANK. We also studied iterative PageRank updating algorithm which works better than Google's PageRank algorithm but there is much more to research and test. We also saw few techniques by which we can improve the calculation of PageRank. This discussion centralizes on Page Ranking scheme when the web data is no longer static but dynamic in nature and poses new challenges for search engines to rank pages.

REFERENCES

[1] Sergey Brin and Lawrence Page, "The Anatomy of a Large-Scale Hypertextual," *Stanford University.*

[2] H. G.-M. L. P. Junghoo Cho, "Efficient Crawling Through URL Ordering," *Stanford University.*

[3] R. K. M. M. E. U. Bahman Bahmani, "PageRank on an Evolving Graph," *IEEE.*

[4] M. P. T. Mandar Kale, "DYNA-RANK: Efficient calculation and updation of PageRank," *IEEE,* 2008.

[5] C. D. M. Amy N. Langville, "Updating PageRank with Iterative Aggregation".

[6] "http://en.wikipedia.org/wiki/Markov_chain," *Wiki.*

[7] C. D. M. By Amy N. Langville, Google's PageRank and Beyond: The Science of Search Engine Rankings.

[8] "PageRank," *Wiki.*

[9] F. Chung, "A Brief Survey of PageRank Algorithms," *IEEE,* 2014.

[10] w. b. M. Sobek, "http://pr.efactory.de/e-pagerank-algorithm.shtml," *eFactory GmbH & Co. KG Internet-Agentur ,* 2002/2003.

APPENDIX

LIST TABLE

LIST OF FIGURES