A look into the Page Rank algorithm

Amit Gupta (amit.gupta@colorado.edu)
Rediate Eshetu(rediate.eshetu@colorado.edu)

1 Introduction

The world wide web (WWW) can be viewed as a library composed of billions of web pages and every day the size is increasing at a tremendous rate. Just like any library however, we need to have an efficient way of organizing and retrieving information. Unfortunately, the WWW is not organized in a traditional library like organization (by author, by title name, by topic etc). This creates a problem for search engines, namely how to efficiently retrieve information in this disorganized and gigantic library. Before Google came into the picture search engines used crude methods for answering search queries. One such system was based on terms. If a web page has all the terms being searched then it will come up in the search results. Furthermore, the frequency of the terms and their location (if a term is on the header then its more important) was important in determining their location on the results page. It is easy to see how malicious users can exploit such search systems for their own benefit.

PageRank was introduced by Larry Page and Sergey Brin and it proposed a radically different way of determining the importance of a web page. [3]. The two key ideas behind PageRank are:

1. A page is deemed important if it has many other pages linking to it (back link). Back links from other important pages counted more than back links from less important pages.

2. The contents of a page were not simply determined by the terms it contains but also by the terms contained on the back links from other pages.

The following formula captures the essence of the importance function of PageRank, if we let the set B_i represent the set of pages linking to page i, the function PR(j) return the PageRank value of j and l_j be the number of links on page j (multiple links to the same page are counted as 1).

$$PR(i) = \sum_{j \in B_i} \frac{PR(j)}{l_j} \tag{1}$$

That is the PageRank value of i is the sum of the PageRank values of all the pages linking to it divided by the number of links on those pages. [3]

These ideas made it very hard for malicious users to manipulate the search results because their importance was in effect not determined by their contents only but also by what other web pages say about them. Below we will detail how exactly page rank determines the importance of all the pages in the WWW.

2 Naive PageRank

The PageRank function assigns a real number to each web page in the WWW (at least the ones discovered so far) and this assigned number determines the importance of a page. The higher the number the more important it is deemed.

The algorithm treats the WWW as a giant directed graph where vertices represent web pages and directed edges represent links. The example below gives an example of an WWW composed of only 3 web pages:

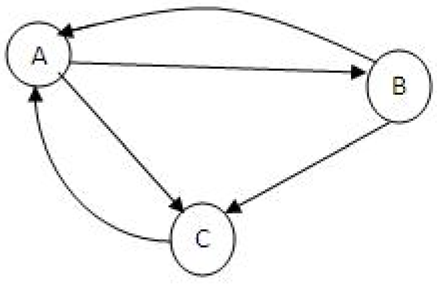

Figure 1: 3 node WWW

This WWW has three web pages A, B, and C. Page A has links to pages B and C, Page B has links to pages A and C and Page C has a link to page A. PageRank uses a transition matrix to represent the information above, if l_i represents the number of links on a page i and the set B_i represents all the web pages that link to webpage i

$$H_{ij} = \begin{cases} 1/l_j & if P_j \in B_i \\ 0 & Otherwise \end{cases} \tag{2}$$

So the WWW in our example will have a transition matrix like this:

$$\begin{bmatrix} 0 & 1/2 & 1 \\ 1/2 & 0 & 0 \\ 1/2 & 1/2 & 0 \end{bmatrix} \tag{3}$$

The rows represent pages A, B and C in that order and the columns represent A, B and C in that order. Therefore, the first column represents the fact that Page A has two links, one to Page B and another to Page C.

3 Random Surfer

Page Rank uses a random surfer model to reason about the matrix. A random surfer follows random links on a web page until he or she gets bored and leaves the WWW. Hence a random surfer that begins at page A in example 1 has a probability of 1/2 to be at page B on the next step , a probability of 1/2 to be at page C and a 0 probability of being at page A. Another random surfer that starts out at page C will have a probability of 1 to be at page A on the next step and 0 probability of being at pages B or C.

Therefore, the PageRank of a page can be thought of as the probability that a random surfer will end up at the page after many iterations.

The probability distribution that the random surfer ends up at a page can be represented by a vector, where the i^{th} entry of the vector represents the probability that the random surfer is at page i after a number of iterations. For example 1, this vector v^0 looks like this initially:

$$\begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix} \tag{4}$$

That is there is equal likelihood that the random surfer starts at any of the pages A, B or C. Notice that if we multiply this vector by the transition matrix we will get the probability of where the random surfer is at after the next iterations $v^1 = M \cdot v^0$

$$v^1 = \begin{bmatrix} 0 & 1/2 & 1 \\ 1/2 & 0 & 0 \\ 1/2 & 1/2 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix} = \begin{bmatrix} 1/2 \\ 1/6 \\ 1/3 \end{bmatrix} \tag{5}$$

So after following one link the random surfer will have a 1/2 probability of being at page A, 1/6 probability of being at page B and 1/3 probability of being at page C. If we repeat this process many times then we are guaranteed the find a vector such that when we multiply the vector by the transition matrix the entries of the vector do not change. That is we will be able to find the eigenvector of the transition matrix related to the eigenvalue 1. That is :

$$v^{k+1} = Mv^k = \lambda \cdot v^k where \lambda = 1 \tag{6}$$

To see this we can keep calculating the next v^1:

$$\begin{bmatrix} 5/12 \\ 1/4 \\ 1/3 \end{bmatrix}, \begin{bmatrix} 11/24 \\ 5/24 \\ 1/3 \end{bmatrix}, ..., \begin{bmatrix} 0.444444 \\ 0.222222 \\ 0.333333 \end{bmatrix} \tag{7}$$

The final vector in effect represents the PageRank value of each page. From that vector we can conclude that A is the most important page in the example WWW, followed by C and B being the least important page. This intuitively makes sense as A and C both have two pages linking to them and B only has one page linking to it. Furthermore A gets all the importance of page C as C only has 1 link and that link is to A. Also note that the values of the final vector sum to 1, hence the Next we will look at why we were able to converge on the final vector.

3.1 Why we converged on the eigenvector

The matrix in example 1 is a column stochastic matrix because all its columns add up to 1. All column stochastic matrices have an eigenvalue of 1, the proof follows.

Proof: Let A be an nxn column stochastic matrix. Note that the transpose of A, A^T is a row stochastic matrix (all rows sum to 1). It is easy to see that a vector v of dimension n consisting of entries of all 1s is an eigenvector for A^T with eigenvalue of 1, that is A^T v=v. Since we know that A^T and A have the same eigenvalues we can conclude that A has an eigenvalue of 1.

Furthermore, we know that the matrix has a unique largest eigenvalue equal to 1. The proof follows: Proof: Let A be an n*n column stochastic matrix and suppose there is an eigenvalue λ such that $A.x = \lambda.x$. Note that since the values of A are non-negative and the columns add to 1, we can assume that the transpose of A, A^T will have rows that add to one. Therefore, the values of the vector $A^T \cdot x$ will not contain a value greater than x_{max}, largest value of x. This is because each element in $A^T \cdot x$ is a convex combination of the values of x. This proves that for $A^T \cdot x = \lambda_1 \cdot x$ its impossible for $\lambda_1 > 1$. Again since λA^T and A have the same eigenvalues we can conclude that $\lambda > 1$ is impossible.

We can assume that the matrix has a set of eigenvalues λ_n such that:

$$|\lambda_1| \geq |\lambda_i|, i = 2, 3, 4, ..., n \tag{8}$$

The eigenvalue λ_1 and the eigenvector corresponding to λ_1 are called the dominant pair. In stochastic matrixes we know $|\lambda_1| = \lambda_1 = 1$. We can assume this because it is easy to see that none of the eigenvalues are negative since no entries in stochastic matrices are negative.

The process of picking a random non-zero approximation of the eigenvector related to the largest eigenvalue and repeatedly multiplying that vector with the matrix until it converges is called the power method. If we assume that the matrix has a dominant eigenvalue then the power method is bound to find the dominant pair of the matrix. Note that our initial vector is a good approximation of the dominant eigenvector because its values sum to 1 (each entry is equal to 1/n where n is the number of pages in the web). The values of the initial vector summing to 1is important because we are indicating that we wish to find the eigenvector of the dominant pair (since there are unlimited number of eigenvectors related to the dominant eigenvalue) that sums to 1 since the PageRank value is a probability of the random surfer ending up at a page after many iterations.

3.2 Are we always bound to get a stochastic matrix?

The short answer is NO! In example 1 we had a graph that had the following properties:

- It was strongly connected, that is there was a path from any node to any other node
- There were no dead-ends, that is there were no nodes that had no links.

Obviously the above properties do not represent the real WWW. In fact the real WWW is found to look like the graph below:

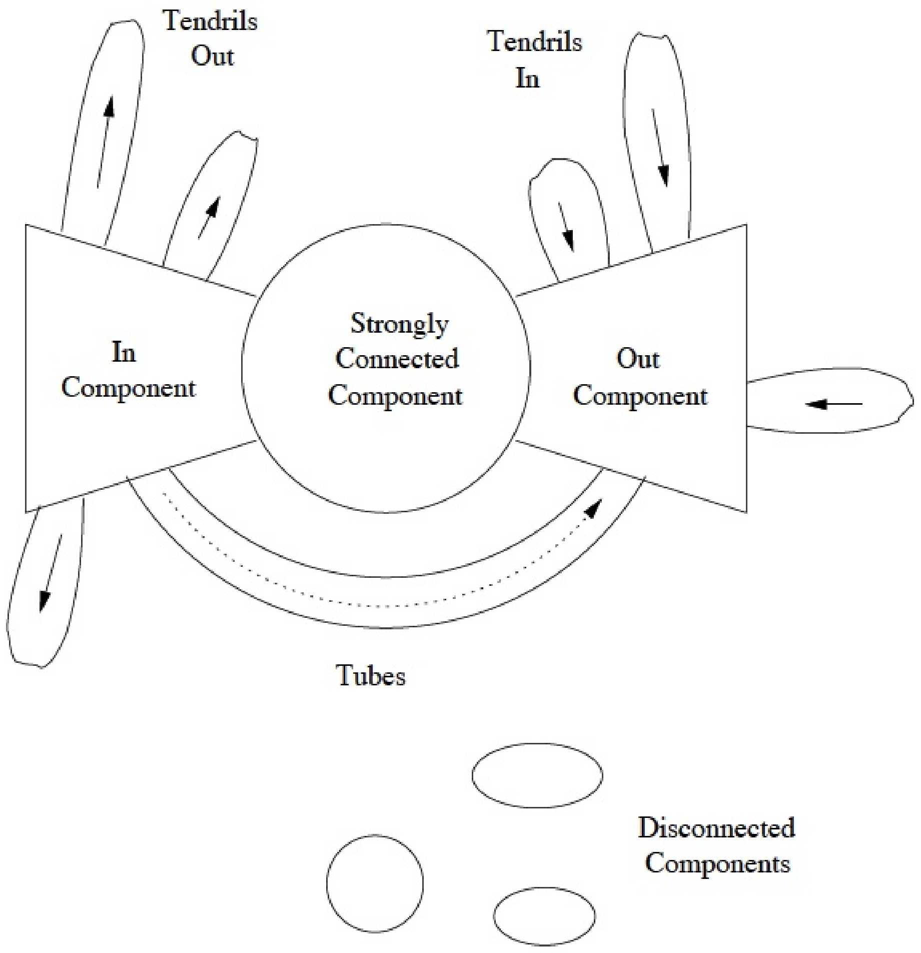

Figure 2: Structure of the WWW

While there is a large strongly connected core to the WWW there are other (almost equally as large) portions of the web that consist of the following (among other):

- In-components: consist of pages that could reach the strongly connected core but cannot be reached from anywhere else. That is ones the random surfer leaves the web pages in the in-components there is no way for him to return.
- Out-componints: consists of pages that could be reached from the

strongly connected core but cannot reach to it. That is ones the random surfer reaches the out-components there is no way for him to escape it.

- Isolated-components: Consists of web pages that are not reachable from anywhere and also cannot reach to anywhere else.
- Tendrils and tubes, other form of groups of web pages not part of the strongly connected core. The presence of several of these structures means that we will not get a stochastic matrix when we represent the real WWW in a matrix. For instance, if there are some dead-end pages with no out links then we will get a column whose value sums to 0 and not 1.

More information at [7]

3.3 Dead Ends

We can modify example 1 to make webpage C a dead-end to see what will happen when we run our PageRank algorithm on it:

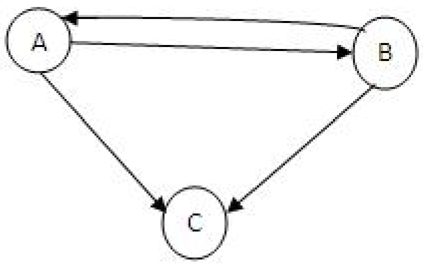

Figure 3: Structure of the WWW

Not that web page C does not have any out links hence it is a dead-end. The following matrix represents the above WWW:

$$\begin{bmatrix} 0 & 1/2 & 1 \\ 1/2 & 0 & 0 \\ 1/2 & 1/2 & 0 \end{bmatrix} \tag{9}$$

If we pick the same initial vector as before:

$$\begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix} \tag{10}$$

We will get the following values for the vectors are each iteration:

$$\begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}, \begin{bmatrix} 1/12 \\ 1/12 \\ 1/3 \end{bmatrix}, \ldots.., \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \tag{11}$$

We see that the final vector is the zero vector. We can think of node C effectively draining all the importance from all the pages linking to it. This is because the random surfer (wherever he may start) is eventually bound to get to node C at which point he cannot escape it. Therefore, eventually the random surfer is bound to be stuck at node C hence the probability he will end up at A or B is zero. Furthermore, from formula 1 we know that the PageRank of a page is fully determined by the PageRank of the pages linking to it, hence the PageRank value of C is also zero.

So we need a mechanism to deal with dead-ends. One method is to delete the dead-end vertices (recursively deleting newly dead-end vertices as well) and their incoming edges. Afterwards, we can run the PageRank algorithm on the resulting graph. Finally, we can compute the PageRank values of the deleted nodes (in the reverse order they were deletes) using formula 1.

Using this method we get the following graph from example 2:

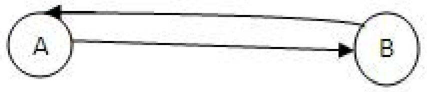

Figure 4: Structure of the WWW

The resulting matrix is simply:

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{12}$$

We choose the initial vector:

$$\begin{bmatrix} 1/2 \\ 1/2 \end{bmatrix} \tag{13}$$

We will get the following values for the vectors are each iteration:

$$\begin{bmatrix}1/2\\1/2\end{bmatrix} \cdot \begin{bmatrix}1/2\\1/2\end{bmatrix} \tag{14}$$

Using formula 1 we can now calculate the PageRank of C:

$$PR(C) = \frac{1}{2} \cdot \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{2} \tag{15}$$

So we have PR(A) = 1/2 , PR(B) = 1/2 and PR(C) = 1/2

Note that the sum of the PageRanks exceeds 1 this is because they no longer represent the actions of a random surfer. However, these values still represent good estimates of the relative importance of the pages. In this case we are determining all these pages have equal importance.

3.4 Teleportation

There is a better way to deal with dead-ends than the above mentioned system and the idea behind this method is teleportation. That is we allow the random surfer to be able to teleport to any page at random whenever he wishes to. This is equivalent to the random surfer following random links on web-pages but at a certain time deciding not to follow any of the links at all but instead decides to type in a random web address on the address bar.

Using teleportation we will also be able to solve the problem of spider traps, strongly connected portion of graphs which are not dead-ends (because every vertex in the node has an out edge) but still are effective dead-ends because once the random surfer gets to any of the vertices in that portion he will never be able to return to where he came from. To illustrate the problem of spider-traps we will modify the graph from the previous example:

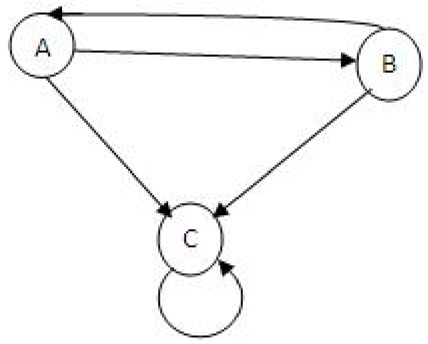

Figure 5: Structure of the WWW

Now C is not a dead-end anymore because it has an out-edge but C represents a spider trap (C can in fact be an arbitrarily large strongly connected sub graph with the property that no vertices in that sub graph connect to A or B).

If we run the PageRank algorithm on the transition matrix for example 4 we get a final vertex:

$$\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{16}$$

The spider-trap C drained all the PageRank from the pages and was able to store it for itself. To model the teleportation we have to modify the iterations in the graph in the following way:

$$v^{k+1} = \beta M v^k + (1-\beta)e/n \tag{17}$$

Where β is a constant (the default value used by Google is 0.85) and e is a vector composed of all 1s. What this equation is saying in effect is: The random surfer will follow one of the links on a page with a β probability or teleport to a random page with a $1-\beta$ probability. During teleportation the random surfer has the same probability of landing at any pages on the web.

Using this new iteration and a value of 4/5 for β we can calculate the PageRank function for example 4:

Transition matrix is now

$$\begin{bmatrix} 0 & 1/2 & 0 \\ 1/2 & 0 & 0 \\ 1/2 & 1/2 & 1 \end{bmatrix} \cdot \frac{4}{5} = \begin{bmatrix} 0 & 4/10 & 0 \\ 4/10 & 0 & 0 \\ 4/10 & 4/10 & 4/5 \end{bmatrix} \tag{18}$$

Our initial vector is as usual

$$\begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix} \tag{19}$$

$$v^1 = \begin{bmatrix} 0 & 4/10 & 0 \\ 4/10 & 0 & 0 \\ 4/10 & 4/10 & 4/5 \end{bmatrix} \cdot \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix} + \begin{bmatrix} 1/15 \\ 1/15 \\ 1/15 \end{bmatrix} = \begin{bmatrix} 1/5 \\ 1/5 \\ 1/5 \end{bmatrix} \tag{20}$$

When we converge we get the following vector:

$$\begin{bmatrix} 0.1111 \\ 0.1111 \\ 0.7777 \end{bmatrix} \tag{21}$$

Intuitively this result makes sense, both A and B only have one incoming links and those links come from relatively unimportant pages (A and B) hence they have very low PageRank values. On the other had the spider-trap C has three incoming links one of which is from a very important page (C).

More information at [2]

4 Efficient Computation

4.1 Representing the Transition Matrix

The web consists of an order of millions (and constantly increasing) number of web pages. Each web page may only link to a handful of other webpages (even considering a generous estimate of say 50 out links per page). These factors cause the Transition Matrix M to be an extremely large and sparse matrix having a lot of zero entries. The storage space required to store an nxn matrix is $O(n^2)$ i.e quadratic in the number of nodes. this is a considerable waste of space as most of these entries wind up being zero. A more efficient way is thus to store only the non-zero elements like an adjacency list. For example:

$$M = \begin{bmatrix} 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix} \tag{22}$$

could be stored as an adjacency list

Source	Out-Degree	Destinations
A	3	B,C,D
B	2	A,D
C	1	A
D	2	B,C

This only requires a storage space proportional to the number of non zero elements in the matrix i,e Linear in number of non zero elements

4.2 MapReduce and Matrix Multiplication

An iteration of PageRank computation as per the equations mention earlier:

$$v^{'} = \beta * M * v + (1 - \beta) * e/n \tag{23}$$

This is essentially a matrix multiplication:

We're trying to compute a product of a square matrix M of dimensions nxn and a column v matrix of dimension nx1 to give a resulting column matrix x of dimension nx1. The i^{th} term of x is given by

$$x_i = \sum_{j=1}^{n} m_{ij} \cdot v_j \tag{24}$$

The challenge in this computation lies in the fact that, in the case of PageRank these vectors (particularly v and x)are often too large to fit into main memory. This is where MapReduce is useful.

MapReduce is a generic distributed execution framework that is essentially a "divide and conquer" approach. It divides the problem input into manageable chunks each of which will be consumed by one of several identical workers (Map Tasks). The results spit out by each of the Map Tasks will be merged with each other (Reduce Tasks) to eventually give the final result. The Map and Reduces tasks are run in parallel across several machines, thereby significantly reducing the run time for an enormous computation:

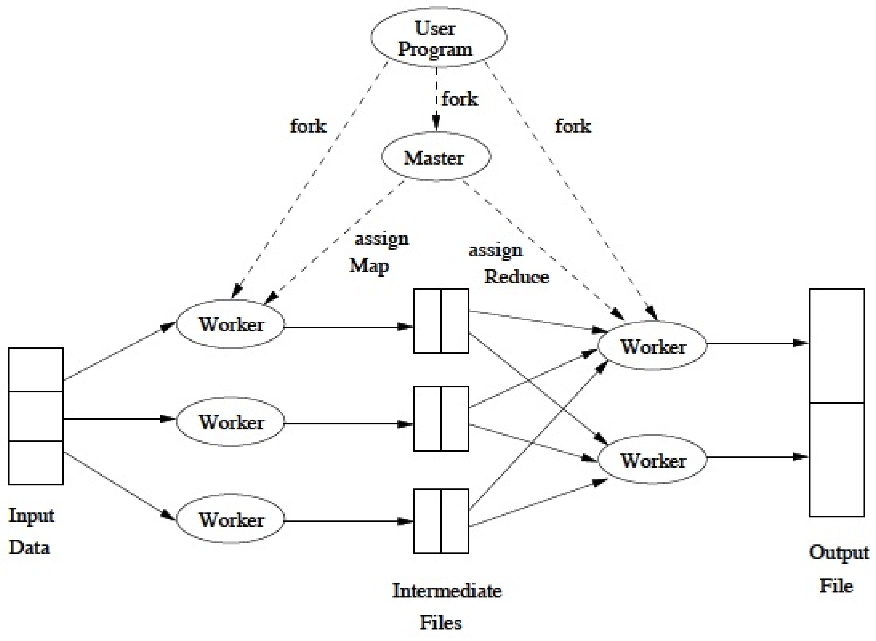

Figure 6: Map Reduce in action

One feature of MapReduce that is of interest is that of ”Combiners”. The fundamental idea being that the reduce function is run on each intermediate chunk of data (output of a Map) as well so that some aggregation is performed within the scope of that chunk. This reduces the amount of data exchanged between the Map and Reduce phases of the system and thereby reduces inter-node traffic.

We’re unable to cover MapReduce in length here as it has several subtle details of its own (including fault tolerant features). There are several excellent resources on the WWW to read up about MapReduce including the original paper [5], [6]

4.3 Partitioning Strategies

4.3.1 M column wise and v row wise

Returning to the challenge of matrix multiplication on vectors too large to fit into main memory. One alternative is to divide M into vertical and v into horizontal stripes.

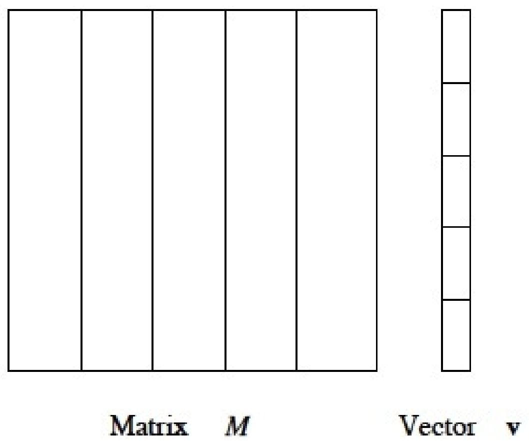

Figure 7: Matrix Column/Row wise Partitioned

The i^{th} stripe of the matrix would only multiply with the i^{th} stripe of the vector. Each map task would get one vertical stripe of M and one horizontal stripe of v. This makes it possible to hold as much of vector v that will fit into main memory.

We should note the following :

- The result vector would be of the same size as vector V. Therefore if V doesn't fit into main memory neither will the result.
- The result vector for page rank becomes the input of the next iteration. Therefore it needs to be partitioned in a similar manner (horizontal stripes)
- Large matrices and vectors like these are often stored in a flattened manner in a file. ex Each vertical stripe of M would be in its own file and each horizontal stripe of V would be in its own file.
- Each of these multiplications contribute their share to each of the terms in the result vector.

As a consequence this method has the disadvantage of causing excessive file i/o significantly affecting the run time of the computation (I/O thrashing)

4.3.2 Block partitioning

This brings us to the solution of using the block partitioning scheme. The matrix M is partitioned into k^2 blocks and the column vectors are partitioned into k horizontal stripes as usual.

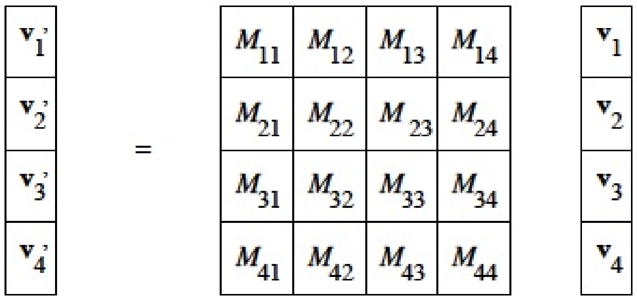

Figure 8: Matrix Block Partitioned, k = 4

Each map task gets a block of the matrix M_{ij} and the corresponding horizontal slice of the vector v_j. Therefore each slice v_j is used by k map jobs (i.e transmitted over the network a maximum of k times). The advantage gained here is that the M_{ij} and v_j only contribute to the i_{th} stripe of the result vector $v_i^{'}$ and no other stripe. Thus, by properly choosing k, both v_j and $v_i^{'}$ can be kep in memory while M_{ij} is processed. The individual components of $v_i^{'}$ will be combined by the reduce stage of execution (as a simple matrix addition).

The second component of the PageRank calculation (i.e taxation) is again a simple addition to each term of the result matrix and therefore can be handled during generation of the final horizontal strips of the result vector $v^{'}$.

4.3.3 Representing the block partitioned Transition Matrix

This does raise a subtle issue over how to represent a transition matrix that is to be portioned block-wise. Using a small modification to our previous representation of a spare transition matrix, we can overcome this situation.

For each block, we need information of all the columns, that have a non zero element within that block. For a given column, we end up repeating the out-degree term for every column that "overlaps" the block and has a non zero entry in that same column. This causes some extra space to be wasted, but this is upper bounded by the out-degree of that node(number of non zero entries in the corresponding column) itself.i.e *The out degree term is repeated a maximum number of times as the out degree itself.* This causes no dramatic increase in the space requirement for storing the transition matrix M.

Lets consider this example 4x4 matrix broken into 4 blocks each of size 2x2:

$$M = \begin{bmatrix} 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix} \tag{25}$$

Our original representation of M:

Source	Out-Degree	Destinations
A	3	B,C,D
B	2	A,D
C	1	A
D	2	B,C

Out representation of the blocks:

M_{11}

Source	Out-Degree	Destinations
A	3	B,
B	2	A

M_{12}

Source	Out-Degree	Destinations
C	1	A
D	2	B

M_{21}

Source	Out-Degree	Destinations
A	3	C,D
B	2	D

M_{22}

Source	Out-Degree	Destinations
D	2	C

5 Runtime, Space complexity

Page Rank calculations are essentially matrix multiplications. Matrix multiplication fundamentally takes $O(n^2)$ time where n is the dimension of the transition matrix M. There is no "magic pill" or shortcut to reduce this asymptotic running time.

What has been leveraged here however is problem partitioning and parallelism. The problem has been so partitioned as to minimize the I/O and keep the working set size in memory as much as possible. And each sub problem has been run in parallel providing what is technically a constant time improvement i.e the same amount of time it takes to run a matrix multiplication with input size of $\frac{n}{k}$, giving us the k^2 factor improvement.

With k chosen to be very large i.e smaller blocks, the constant factor provides very significant improvement to the point of making an infeasible calculation not only feasible but fast. The price however is paid in the quantity of hardware resources required to run this computation.

More information at [4],

6 Topic Sensitive PageRank

There are several improvements that one could make to PageRank. One such is to make it topic sensitive. There are several keywords that are found in pages of vastly different topics of interest. For ex. the word "jaguar" could be found on pages related to wildlife and also on pages related to automobiles. Search quality could be improved if each user had his personalized PageRank vector based on his/her topics of interest. This is obviously not feasible and scalable as there are billions of users and growing.

The idea is to create very wide buckets of topics (ex sports, politics etc) and have a page rank vector for each of these very broad categories. For each user, a smaller vector could be stored of his/her relative interest in these topic categories and the appropriate page rank vector chosen to produce search results. This is more scalable and feasible but is however not a "sharp tool".

7 Biased Random Walks

Extending the idea of random surfers and combining it with the goal of topic sensitive PageRank vectors. There is an intuition that a group of pages that are about a certain common topic would tend to link to each other more so than others. i.e a page that links to a page about music would tend to be about music too.

Therefore if we modify the way in which we introduce random surfers into the mix, by letting them teleport to pages of a chosen topic, rather than any random page, we could converge on more relevant rankings. i.e the page rank iteration calculations now become:

$$v^{'} = \beta * M * v + (1 - \beta) * e_s/n \tag{26}$$

where e_s is the teleport set instead of the unit vector, which contains 1 for a page that belongs to the desired topic and 0 for a page that doesn't. The computation itself is carried out in a similar manner as the "vanilla" PageRank.

8 Link Spam

Link spam is a way to fool a naive PageRank calculation. If one were to create a lot of links pointing to a target page, PageRank could be made to believe that the page being pointed to is important. This is called link spamming.

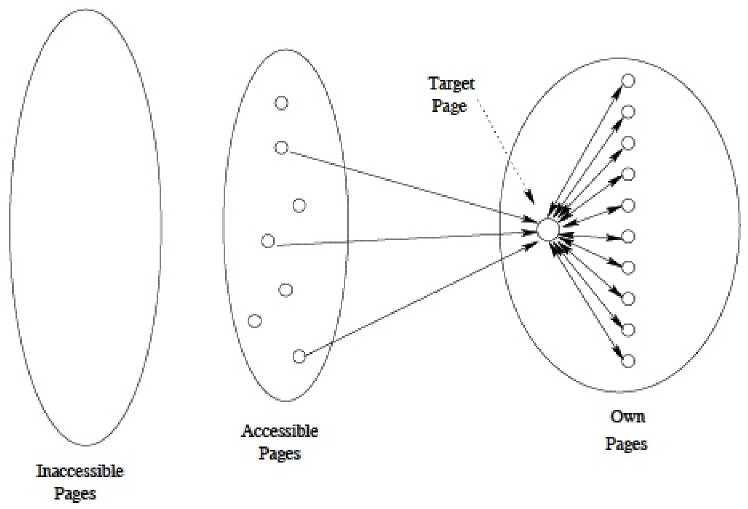

Figure 9: An example spam farm

There are 3 kinds of sites from the perspective of a link spammer:

- Pages not accessible.
- Pages accessible but not owned directly.(ex blog comment fields, discussion forums, online mailing list archives etc)
- Pages owned and therefore accessible.

A link spammers effort is to create as many references to the page he/she want to gain significance in page rank. Such references are called spam farms.

9 Trust Rank, Panda & Penguin

There are several techniques that search engines use to defeat the effort of link spammers:

- The compile a list of controlled domain pages (ex. .gov, .edu etc) that a deemed "trustworthy" i.e less likely to be participants in a spam farm. They then create a page rank vector similar to topic sensitive page rank where essentially the "topic" is a list of trustworthy pages. They call this version of page rank TrustRank.
- Webcrawlers try to identify patterns in the links of spam farms. They try to assign the participant pages a low page rank to being with thereby the target page only receives smaller shares of already small page ranks. This is however a problem that is constantly adapted to as the number of patterns of a spam farm can be innumerable.

Increasingly search engines have damped down the importance of PageRank as the sole measure of importance of a page. Google uses about 250 factors (only one of which is PageRank), with different weights, that go into deciding the relative importance of a page. Other plausible factors are the speed of the website, age of the page (new is better) and so on and so forth. Bing claims it uses about 1000 factors. It is also claimed by both companies that these factors themselves constantly change as well as their relative weights. These are of course closely guarded business secrets and always "adaptive moving targets".

As an example of 2 other such constantly changing factors introduced by Google:

- Panda: Released by Google in February 2011, a new metric input to their page ranking algorithm. It is a machine learning algorithm that attempts to look for similarities between pages of higher quality and lower quality.

- Penguin: Released as recent as April 24th 2012 (a week ago at the time of this writing), designed to target pages and lower their overall rank of the ones it believes are link spamming. [1]

[7]

References

[1] Penguin. 2012. `http://searchengineland.com/penguin-update-recovery-tips-advice-119650`.

[2] David Austin. How google finds your needle in the web's haystack. Technical report, AMS. `http://www.ams.org/samplings/feature-column/fcarc-pagerank`.

[3] Page Brin. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford. `http://ilpubs.stanford.edu:8090/422/`.

[4] Taher Haveliwala. Efficient computation of pagerank. Technical report, Stanford, 1999. `http://ilpubs.stanford.edu:8090/386/`.

[5] Sanjay Ghemawat Jeffrey Dean. Mapreduce: Simplified data processing on large clusters. 2004. `http://research.google.com/archive/mapreduce.html`.

[6] Jeff Ullman and Anand Rajaraman. *Chapter 2 of Mining of Massive Datasets*. Cambridge UK, 2010.

[7] Jeff Ullman and Anand Rajaraman. *Chapter 5 of Mining of Massive Datasets*. Cambridge UK, 2010.