PageRank

Jonathan Freire*, Samantha Naranjo†, Nicolás Serrano*, Cristina Sabando †, and Johana Telenchana †

Abstract.
The PageRank Algorithm is one of the most iconic milestones on Internet development becauses it takes advantage of the existing web network and categories by relevance each webpage about an specific topic. In this Advanced Algebra course task, we have made a compact bibliographic review of the topic and a simple algorithm implementation in C that can be use to understand the PageRank performance. The conclusion remarks the importance of computation of eigenvalues (by the Power Method) in real life applications like Web Search Engines.

Key words.
PageRank, Web Search, Google Search Engine, Eigenvalues and Power Method.

1. Introduction.

With the constant growth of the World Wide Web, new challenges regarding information retrieval surface everyday. It is not a surprise that the web is heterogeneous and very large. In fact, there are more than 1.8 billion websites on the world wide web as of April of 2018. Of these, less than 200 million are active[3]. Since the early stages, search engines have developed different methods to rank web pages. Until today, the occurrence of a search phrase within a document is one major factor within ranking techniques of virtually any search engine. To optimize search results,the concept of link popularity was developed to avoid good rankings for pages which are not significant and, in some cases, deceitful. Following this concept, the number of inbound links for a document measures its general importance. Hence, a web page is generally more important, if many other web pages link to it.
The PageRank algorithm, which would later become the first prototype of Google's web search engine, was first introduced on a 1998 publication by Stanford University graduates, Larry Page and Sergey Brin. Contrary to the concept of link popularity, PageRank's basic approach is that a document is considered more important the more other documents link to it. So, within the PageRank concept, the rank of a document is given by the rank of those documents which link to it. Their rank again is given by the rank of documents which link to them. Therefore, one can define the PageRank of a documente recursively by the PageRank of other documents.

In their publications, Page and Brin state that they consider PageRank as a model of user behavior, where a random surfer clicks on links at random, not caring about content.The random surfer visits a web page with a certain probability which derives from the page's PageRank. The probability that the random surfer clicks on one link is solely given by the number of links on that page. So, the probability for the random surfer reaching one page is the

*Department of Computer Sciences and Engineering, Yachay Tech. ID (jonathan.freire@yachaytech.edu.ec, nicolas.serrano@yachaytech.edu.ec).
†Department of Mathematics, Yachay Tech. ID (samantha.naranjo@yachaytech.edu.ec, maria.sabando@yachaytech.edu.ec, johana.telenchana@yachaytech.edu.ec).

sum of probabilities for the random surfer following links to this page. Now, this probability is reduced by the damping factor d, which is, depending on the degree of probability therefore, set between 0 and 1. The higher d is, the more likely will the random surfer keep clicking links. Since the surfer jumps to another page at random after he stopped clicking links, the probability therefore is implemented as a constant (1-d) into the algorithm.

2. PageRank Method. Link analysis ranking method consider the web as a directed graph G where each node represents pages. This method computes the rank of every page based on the hyperlink structure of the web, and not on the basis of the content of the pages [1]. Thus, PageRank concept is that a page is important if it is pointed by other pages likewise important. Also, PageRank is an iterative computation method for assigning a rank score to every page in the web [1]. This method compute a principal eigenvector using the power method.

The PageRanks process consists in the following [2]:

- Let N_u be the outdegree of page u nad let $Rank(p)$ represent the importance of page p (i.e., PageRank). The the lin (u, v) confers $Rank(u)/N_u$ units of rank to v

This basic idea leads to the following iterative fixpoint computation that yields the rank vector $\vec{Rank}$ over all of the pages on the web.

- If n is the number of the pages, assign all pages the inicial value $1/n$. Let B_u represent the set of pages pointing to v. In each iteration, propagate the ranks as follows:

$$\forall_u Rank^{(i+1)}(v) = \sum_{u \in B_u} Rank^{(i)}(u)/N_u$$

For $u \in B_u$, the edge (u, v) guarantee that $N_u \geq 1$ [2]

The iterations continue until $\vec{Rank}$ stabilizes to within some threshold. Thus, the final vector contain the PageRank Vector over the web. Also, this vector is computed only once after each crawl of the web and the values can be used to influence the ranking search result.

The process can also be expressed as the following eigenvector calculation:

a) Let M be a square, stochastic matrix corresponding to the directed Web graph. If there is a link from page j to page i, let the matrix entry m_{ij} have the value $1/N_j$, where N_j is the number of forward links, i.e. the number of pages that j points to. m_{ij} represents the probability to visit page j starting from page i.

b) Let all other entries have the value 0.The diagonal entries of the matrix usually are zero because a page has no links to itself.

c) Let $\vec{r_0} = 1/n$ be an initial vector, where n is the number of web sites (or the initial rank assigned to every page).
The rank of a page j is found by using the following iterative formula:

$$R(j) = d \sum_{i=1}^{n} R(i)/N_i,$$
$$\text{The approximation factor is } d \approx 0.15$$

d) One iteration of the previous fixpoint computation corresponds to the matrix-vector multiplication $M \times \vec{r}$ (where $\vec{r}$ is the rank vector over all the pages on the web).
Since the Power method is being used, repeating multiplying $\vec{r}$ by M yields the dominant eigenvector $\vec{r}$ of the matrix M.
In other words, $\vec{r}$ is solution to :

$$\vec{r} = M \times \vec{r}$$

This eigenvector, which is associated to the largest eigenvalue, corresponds to the most important page.

Because M^T is the stochastic transition matrix over the graph G, Page Rank can be viewed as the stationary probability distribution for the Markov chain induced by a rand walk on the Web graph G [2].

The problem that arise with the PageRank method is that its convergence is only guarantee if M is irreducible and periodic. The latter is guarantee in practice for the wed, while the former is true if:

1) We add a complete set of outgoing edges to nodes in G with outdegree 0,
2) Damp the rank propagation by a factor $1 - \alpha$ by adding a complete set of outgoing edges, with weight α/n, to all nodes.

On the other hand, we can accomplish this task by constructing a new matrix M in the following way:

1) Let $\vec{p}$ be the $n-$dimensional column vector representing a uniform probability distribution over all nodes:
$$\vec{p} = \left[\frac{1}{n}\right]_{nx1}$$
2) Let $\vec{d}$ be the $n-$dimensional column vector identifying the nodes with outdegree 0:
$$d_i = \begin{cases} 1 & if - deg(i) = 0 \\ 0 & otherwise \end{cases}$$
3) Then, we can construct M' as follows:
$$M' = (1 - \alpha)(M + D) + \alpha E.$$
Where D is the cross product between $\vec{p}$ x $\vec{d_i}$ and E represents the cross product between $\vec{p}x$

This modification improves the quality of PageRank by introducing a decay factor $(1 - \alpha)$ which limits effect of rank sinks, in addition to guaranteeing convergence to a unique vector [2]. With the matrix M', we can express PageRank as the solution

$$\vec{r} = \text{M'} * \vec{r}$$

By appropriately selecting $\vec{p}$, the rank vector can be made to prefer cetain categories of pages. The bias factor α specifies the degree to which the computation is biased towards $\vec{p}$ [2].

3. Algorithm Implementation.

To implement and test the PageRank method, we have written a code in C and we use a text file document with all the information of our web example. The following subsections explains more about the code functionality.

3.1. Text file.

By now, we have discussed that PageRank algorithm analyzes a web system using the links from page to page. Each line in the text file (web.txt) represents a link, the first number tell us from which page we are moving and the second number tell us to the destiny node. For example:

2 3
3 4
5 6

means that there is a link from page 2 to page 3, a link from page 3 to page 4 and a link from page 5 to page 6.

In order to point the file, we will use a pointer FILE with name fp.

```
FILE *fp;
```

3.2. Collecting data from web.txt. First, we need the number of pages we are analyzing. Remember that in a matrix representation the first webpage is called "0" so to get the right ammount of pages we have to add 1 to the largest value found in the nodefrom and nodeto columns. We are using a while loop that scans the file and compares the respective values.

To end the while loop we need a variable (ch) that notify us when we have reach the End of the File (EOF).

```
int n=0;
int ch,nodef,nodet;

fp=fopen("nico.txt","r");
while ((ch = fgetc(fp)) != EOF)
{
     fscanf(fp,"%d%d", &nodef, &nodet);
          if(nodef >= n)
       n=nodef + 1;
     else if (nodet >= n)
       n=nodet + 1;
}
printf("La matriz ser de %d X %d \n\n", n,n);
fclose(fp);
```

Now that we have the size of the matrix, we create a A matrix (nxn) and initialize all elements to zero. After that, we scan the file again but this time, we set the elements to one if there exist a node between the pages represented by column and row of the matrix.

```
float a[n][n];
  for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
      a[i][j] = 0.0;

int nodefrom, nodeto;
fp = fopen ("nico.txt", "r");
while (!feof (fp))
 {
  fscanf (fp, "%d%d", &nodefrom, &nodeto);
  a[nodefrom][nodeto] = 1.0;
 }
  fclose (fp);
```

3.3. Creating OutLink Vector, Stochastic A and A Transposed. Until now, the matrix A can tell us where is a link between a page and another but we need to know the probability of going to a specific page if we are standing on another page.

To do this, we create the OutLink vector, that tell us how many links go out from a specific page.

```
int out_link[n];
// initialize vector to zero
for (int i = 0; i < n; i++)
  out_link[i] = 0;

//out_link vector indicates how many links go out from page [row]
for (int i = 0; i < n; i++)
{
  for (int j = 0; j < n; j++)
      {
        if (a[i][j] != 0.0)
        {
            out_link[i] = out_link[i] + 1;
        }
      }
}
```

After that, we divide each A[i][j] by OutLink[i]. Some webpages are PDF files or no out link pages so they cannot be divide by OutLink[i] = 0. In this case, we put the probability of going to that webpages with respect to all pages in the web.

```
for (int i = 0; i < n; i++)
{
  if (out_link[i] == 0)
      {
        for (int j = 0; j < n; j++)
          a[i][j] = 1.0 / n;
      }
  else
      {
        for (int j = 0; j < n; j++)
          a[i][j] = a[i][j] / out_link[i];
      }
}
```

For the power method to converge we need to use the transposed stochastic matrix A. This means chaging the rows for the columns an viceversa. To avoid getting some data delete, we opt to create a new matrix called "at" that stands for "A Transposed".

```
float at[n][n];
for (int i = 0; i < n; i++)
  {
```

```
        for (int j = 0; j < n; j++)
            {
                at[i][j] = a[j][i];
            }
    }
```

3.4. Setting up initial parameters and vectors. The PageRank algorithm uses a damping factor (d) that represents the reliability of the results. This factor depends on the size of the example, the correctness of the web graph and the presence of sub-webs inside the analyzed web. This value has to be inside the interval [0 - 1]. The original PageRank article set the damping factor equal to 0.85.

The looping variable acts like a switch in the PageRank loop to stop it and the k variable counts the number of iterations done until the end of the process.

```
float d = 0.85;
int looping = 1;
int k = 0;
```

The "Random Surfer" vector represents the probability of the user to start in a certain webpage. In context, lets imagine we just started Google Chrome and we want to start navigating in the web. We have no links in our browser and only the search bar. There is the change of going to any website we want. To represent this we use the "Random Surfer" vector initialized with all elements' values equal to 1 / n.

```
float x[n];
for (int i = 0; i < n; i++)
  {
      x[i] = 1.0 / n;
  }
```

To finish the set up, we create the resulting vector called b[n].

```
float b[n];
```

3.5. PageRank Loop. Lets imagine a system Ax = b. Right now we have all elements ready (Stochastic Transposed Matrix A, the initial Random Surfer vector and the resulting vector b).

The loop starts by giving b[n] elements the value zero. After that we multiply Ax and store the result inside the vector b[n]. Then we use the damping factor and the PageRank formula.

We create a error variable and add it up with the absolute value of the difference between each element of x[n] and b[n]. If the error is less that our tolerance (in our case 0.000001) the looping variable will be set to zero and the PageRank loop will stop in the next step.

To finish, we update x[n] with the new b[n] vector and count the iteration by adding up one to k.

```
while (looping)
  {

  for (int i = 0; i < n; i++)
      {
        b[i] = 0.0;
      }

  for (int i = 0; i < n; i++)
      {
        for (int j = 0; j < n; j++)
          {
            b[i] = b[i] + (at[i][j] * x[j]);
          }
      }

  for (int i = 0; i < n; i++)
      {
        b[i] = d * b[i] + (1.0 - d) / n;
      }

  float error = 0.0;
  for (int i = 0; i < n; i++)
      {
        error = error + fabs (b[i] - x[i]);
      }

  if (error < 0.000001)
      {
        looping = 0;
      }

  for (int i = 0; i < n; i++)
      {
        x[i] = b[i];
      }

  k = k + 1;
  }
```

The final vector x[n] will be the EigenVector and the element with the biggest value (Eigenvalue) will be the one that represents the most important page in the web system provided.

4. Conclusions. It can be concluded that the calculation of eigenvalues and eigenvectors of a matrix is very reliable because they help us to know the most relevant characteristics of a data set, in this case, thanks to the algorithm, it allowed us to know how to index a set of data to determine the importance of each data.

The use of PageRank Algorithm consists of several steps, in first place a matrix is built based on the nodes of each page, for this we have as input a node and the node it points to, then we obtain an inverse matrix for the next step where the formula of the Power Method is applied for a determined number of iterations, after performing this procedure the resulting vector will have the values of the PageRank of each node. Currently this algorithm can have several real applications in many fields and not only in the indexing of web pages that was originally created for this program.

This method is very helpful because the logic on which it is based makes much more sense when it comes to measuring the popularity of a web page, because it compares them as if it were a person's popularity.

REFERENCES

[1] R. G. Atul Kumar Srivastava and P. K. Mishra, *Analysis of iterative methods in pagerank computation*, Journal of Information and Optimization Sciences, 38 (2017), pp. 1–14, https://doi.org/10.1080/02522667.2017.1372914.

[2] T. H. Haveliwala, *Topic-sensitive pagerank: A context-sensitive ranking algorithm for web search*, IEEE Computer Society, 15 (2003), pp. 784–796, https://doi.org/10.1109/TKDE.2003.1208999.

[3] *Internet live stats,* Total Number of Websites.(n.d), *year=2018, month=april, url = http://www.internetlivestats.com/total-number-of-websites/* .