

Page Ranking Implemented in Java

By Nima Goodarzi, October, 24, 2009

Introduction

Within the past few years, Google has become the far most utilized search engine worldwide. This success is because of the high quality results in comparison to other search engines. This high quality result is because of the Page Ranking feature that google is using.

Each document is assigned with a ranking while it is being indexed. when you search for a key word, pages with higher page rank are displayed on top.

The page rank algorithm is found by Google founders Lawrence Page and Sergey Brin.

In this article we first go through the algorithm and then implement it in Java.

The PageRank Algorithm

The original PageRank algorithm was described by Lawrence Page and Sergey Brin in several publications. It is given by

$$PR(A) = (1-d) + d (PR(T1)/C(T1) + \dots + PR(Tn)/C(Tn))$$

where

- $PR(A)$ is the PageRank of page A,
- $PR(Ti)$ is the PageRank of pages Ti which link to page A,
- $C(Ti)$ is the number of outbound links on page Ti and
- d is a damping factor which can be set between 0 and 1 usually 0.85.

So, first of all, we see that PageRank does not rank web sites as a whole, but is determined for each page individually. Further, the PageRank of page A is recursively defined by the PageRanks of those pages which link to page A.

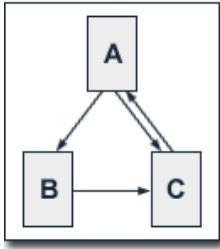
The PageRank of pages Ti which link to page A does not influence the PageRank of page A uniformly. Within the PageRank algorithm, the PageRank of a page T is always weighted by the number of outbound links $C(T)$ on page T. This means that the more outbound links a page T has, the less will page A benefit from a link to it on page T.

The weighted PageRank of pages Ti is then added up. The outcome of this is that an additional inbound link for page A will always increase page A's PageRank.

Finally, the sum of the weighted PageRanks of all pages Ti is multiplied with a damping factor d which can be set between 0 and 1. Thereby, the extend of PageRank benefit for a page by another page linking to it is reduced.

Example

The characteristics of PageRank shall be illustrated by a small example.



We regard a small web consisting of three pages A, B and C, whereby page A links to the pages B and C, page B links to page C and page C links to page A. According to Page and Brin, the damping factor d is usually set to 0.85, but to keep the calculation simple we set it to 0.5. The exact value of the damping factor d admittedly has effects on PageRank, but it does not influence the fundamental principles of PageRank. So, we get the following equations for the PageRank calculation:

$$PR(A) = 0.5 + 0.5 PR(C)$$

$$PR(B) = 0.5 + 0.5 (PR(A) / 2)$$

$$PR(C) = 0.5 + 0.5 (PR(A) / 2 + PR(B))$$

These equations can easily be solved. We get the following PageRank values for the single pages:

$$PR(A) = 14/13 = 1.07692308$$

$$PR(B) = 10/13 = 0.76923077$$

$$PR(C) = 15/13 = 1.15384615$$

It is obvious that the sum of all pages' PageRanks is 3 and thus equals the total number of web pages. As shown above this is not a specific result for our simple example.

For our simple three-page example it is easy to solve the according equation system to determine PageRank values. In practice, the web consists of billions of documents and it is not possible to find a solution by inspection.

Implementation

Now we are going to implement the Page Rank algorithm and find the page rank of the pages in the given example.

One of the most important parts of this implementation is to solve the linear equation. As you can see in the example, for a three pages scenario we need to solve a 3*3 linear equation, so if we have a collection of n pages we need to solve an $n*n$ linear equation.

To solve the linear equation, I used JAMA package which is available from:

<http://math.nist.gov/javanumerics/jama/>

Now all we need to do is to find the linear equations and use the JAMA package to solve them and find the page ranking of the pages.

As you can see, the linear equations of the above example are:

$$PR(A) = 0.5 + 0.5 PR(C)$$

$$PR(B) = 0.5 + 0.5 (PR(A) / 2)$$

$$PR(C) = 0.5 + 0.5 (PR(A) / 2 + PR(B))$$

By a simple replacement we have :

$$PR(A) - 0.5 PR(C) = 0.5$$

$$- 0.25 PR(A) + PR(B) = 0.5$$

$$- 0.25 PR(A) - 0.5PR(B) + PR(C) = 0.5$$

By using the matrix theorem we have :

$$\begin{bmatrix} 1 & 0 & -0.5 \\ -0.25 & 1 & 0 \\ -0.25 & -0.5 & 1 \end{bmatrix} \begin{bmatrix} PR(A) \\ PR(B) \\ PR(C) \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.5 \\ 0.5 \end{bmatrix}$$

As the first step, we have to find the parameters in this equation. Parameters are the page ranks of the related collection, for example in this example we have three parameters which are PR(A), PR(B) and PR(C).

To find the parameters we start from a page and check the inbound links of it, then we do same for the inbound links. We repeat this action until all the related pages are found.

```
private void generateParamList(String pagelId){
    // Add the starting page.
    if(!params.contains(pagelId))
        params.add(pagelId);

    // Get list of the inbound pages
    String[] inc = getInboundLinks(pagelId);

    // Add the inbound links to the params list and do same for inbound links
    for(int i = 0; i < inc.length; i++){
        if(!params.contains(inc[i]))
            generateParamList(inc[i]);
    }
}
```

Now we have the parameters of the linear equation, the next step is to find the constants of these parameters.

```
/*
 * This method returns the constant of the given variable in the linear equation.
 */
private double getMultiFactor(String sourceId, String linkId){
    if(sourceId.equals(linkId))
        return 1;
    else{
        String[] inc = getInboundLinks(sourceId);
        for(int i = 0; i < inc.length; i++){
            if(inc[i].equals(linkId)){
                return -1 * (DAMPING_FACTOR / getOutboundLinks(linkId).length);
            }
        }
    }
    return 0;
}
```

Assume that we are in page A, the constant of the page ranks are following this rule:

- Page rank of a, $PR(A)$, is 1
- If there is no inbound link from B to A, page rank of B, $PR(B)$, is 0;
- If there is an inbound link from B to A, page rank of B, $PR(B)$, is equal to (damping factor / number of the outbound links from B). But as we are moving this phrase to the other side of the equation we multiply it by -1.

Now we have the equation parameters and the constants, the next step is to generate the matrix based on these information.

```
/*
 * This method generates the matrix of the linear equations.
 * The generated matrix is a n*n matrix where n is number of the related pages.
 */
private double[][] generateMatrix(){
    double[][] arr = new double[params.size()][params.size()];
    for(int i = 0; i < params.size(); i++){
        for(int j = 0; j < params.size(); j++){
            arr[i][j] = getMultiFactor((String) params.get(i), (String) params.get(j));
        }
    }
    return arr;
}
```

In this method we iterate over the parameter list and add the constants of the equations in a 2 dimensional array. The first dimension is regards to the equation and the second is for the parameters constants.

We need one more method to create the Matrix object and solve the equation. As mentioned before, we use JAMA package to solve this equation.

```
/*
 * Solve the equation of ax=b, which :
 * a is the generated matrix based on the parameter constants.
 * x is the page ranks matrix.
 * b is a n*1 matrix which all the values are equal to the damping factor.
 */
public double rank(String pageId){
    generateParamList(pageId);

    Matrix a = new Matrix(generateMatrix());

    double[][] arrB = new double[params.size()][1];
    for(int i = 0; i < params.size(); i++){
        arrB[i][0] = 1 - DAMPING_FACTOR;
    }

    Matrix b = new Matrix(arrB);

    // Solve the equation and get the page ranks
    Matrix x = a.solve(b);

    // Find the index of the requested page in the parameter list.
    int ind = 0;
    int cnt = 0;
    for(Iterator it = params.iterator(); it.hasNext();){
        String curPage = (String) it.next();
        if(curPage.equals(pageId))
            ind = cnt;
        cnt++;
    }

    // Return the page rank
    return x.getArray()[ind][0];
}
```

The complete source code is available from <http://www.javadev.org/files/ranking.zip>