The PageRank Algorithm

Ritwik Saha
B.Tech in Electrical Engineering
Indian Institute of Technology, Mandi
Mandi, India
b16110@students.iitmandi.ac.in

***Abstract*—In this report we would start off by discussing the history of PageRank, how it spawned a multi-billion dollar company and the recent advances in the algorithm. Then we would take a deep dive into the technicalities and mathematics of pagerank, and why it is something that comes in the realm of big data. Also, we would discuss the MapReduce implementation on Hadoop and the discuss the challenges in detail encountered while executing the implementation.**

I. History of PageRank

As a part of their reserach project Stanford duo Larry Page and Sergey Brin developed an algorithm for a new kind of search engine, and aptly named it PageRank, probably after Larry Page. It was based upon a simple idea but it made google the holy grail of search engines. The idea was to give a page a higher ranking based on two factors,

- A page to which more pages link must rank high,
- and pages with high importance link to other pages of high importance

Although, this aproach was never used before in ranking internet webpages, such approaches were suggested by various thinkers and scientists in other domains of life. We know that earliest forerunners of PageRank can be attributed to a Harvard economist Wassily Leontief, who developed a way to rank various sectors of US economy by the importance of what supplies it. Similar approaches were used to model importance of certain people in social hierarchy, which was based on the importance of people who endorsed that person.

Terry Winograd and Rajeev Motwani were co-authors along with Page and Brin in the first paper about the idea and laying out the initial prototype. Shortly after, Google Inc. was founded by Page and Brin. While page ranking algorithm has diversified and takes into account many parameters, PageRank remains one of the main basis of Google's search engine.

A. Search Engine Optmization

Emergence of PageRank as a deciding factor in determining relevance of a link in Search Engine gave rise to a new field in industry, Search Engine Optimization or SEO in short. Search Engine Optimization is the process of maximizing the number of visitors to a particular website by ensuring that the site appears high on the list of results returned by a search engine.

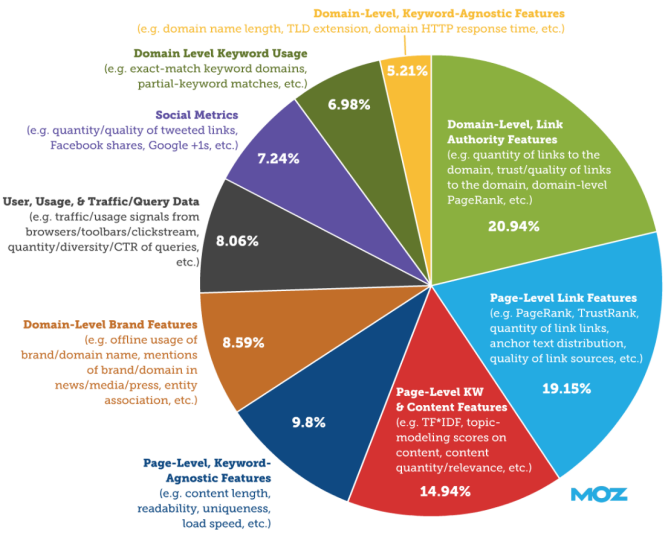

Fig. 1. Weighting of Thematic Clusters of Ranking in Google

B. Google Toolbar

Google Toolbar was an extension released by Google for many different browsers which rated any intenet webpage on a scale of 0 to 10, 10 being the most popular according to the PageRank algorithm. However, in March of 2016, the feature was withdrawn and the underlying API ceased to operate.

978-1-7281-4387-3/19/$31.00 © 2019 IEEE

II. The Algorithm

A. The Simplified Algorithm

Let us assume a small universe of four pages as shown below. If there is an arrow from Page **A** to Page **B**, that means that there exists an outlink on **A**, that links to **B**.

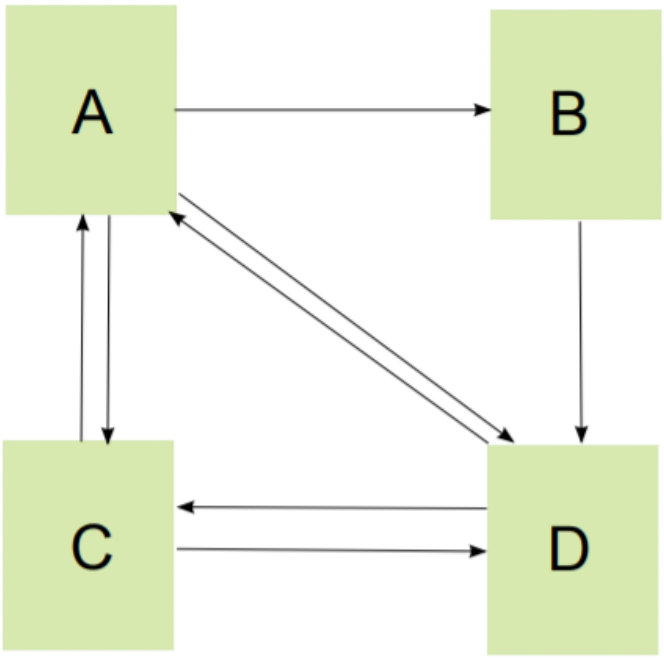

Fig. 2. Universe of 4 pages

We need the probablity with which a random surfer on the internet would land on any of these pages. The simple formula to calculate PageRank is given by :

$$P_i = \sum_j \frac{P_j}{L_j} \tag{1}$$

where j are the pages which have outlinks to page i, P_k denotes PageRank of k and L_k denotes number of outbound links from k. By this defination formula for PageRank of **D** turn out to be :

$$P_D = \frac{P_A}{3} + \frac{P_B}{1} + \frac{P_C}{2} \tag{2}$$

To calculate the PageRank of each node/page, we initially assume some value for each of the page. Conventionally, this probablity is taken to be $\frac{1}{N}$ where N is the number of pages in the universe. Then we iteratively apply the above eqaution, until the PageRanks stop changing considerably from one iteartion to another, that is the answer converges.

B. Formalization of the Algorithm

We create a $N \times N$ matrix with the following value :

$$a_{ij} = \begin{cases} \frac{1}{L_j} & \text{If there is a link from } j \text{ to } i, \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

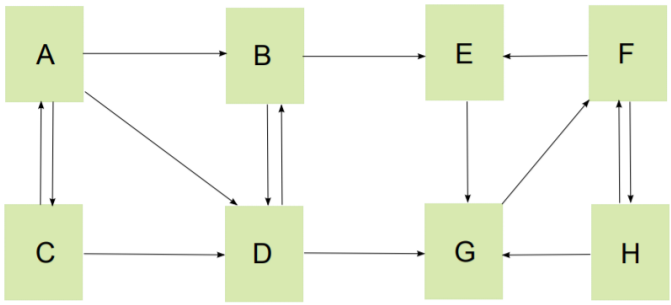

Fig. 3. A reducible graph containing 6 nodes

For the above universe desrcibed in Figure 2 , the matrix **A** turns out to be :

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{3} & 0 & 0 & 0 \\ \frac{1}{3} & 0 & 0 & \frac{1}{2} \\ \frac{1}{3} & 1 & \frac{1}{2} & 0 \end{bmatrix} \tag{4}$$

We initialize a vector **x** with all components $\frac{1}{N}$ where N is the number of nodes, and then keep on multiplying **x** with **A** until the answer converges.

$$\mathbf{x^{i+1}} = \mathbf{A}\mathbf{x^{i}} \tag{5}$$

Running a few iterations on the example universe depicted in Figure 2 we can see that the answer starts to converge.

TABLE I
RUNNING ITERATIONS ON EXAMPLE UNIVERSE

Iteration #	$\mathbf{x_A}$	$\mathbf{x_B}$	$\mathbf{x_C}$	$\mathbf{x_D}$
1	0.25	0.25	0.25	0.25
2	0.25	0.08333333	0.20833333	0.45833333
3	0.33333333	0.08333333	0.3125	0.27083333
4	0.29166667	0.11111111	0.24652778	0.35069444
5	0.29861111	0.09722222	0.27256944	0.33159722
6	0.30208333	0.09953704	0.26533565	0.33304398

1) Dangling Nodes: A node is called a dangling node if it does not contain any out-going link, i.e., if the out-degree is zero.

Suppose, that in the above example **D** has no outlinks, then the resultant matrix would not be column stochastic, i.e., all the elements in a column do not sum to 1. In this case, the pagerank vector unlimately converges to $\vec{0}$.

This problem is tackled by defining a new matrix **A** given by :

$$a_{ij} = \begin{cases} \frac{1}{L_j} & \text{If there is a link from } j \text{ to } i, \\ \frac{1}{N} & \text{If node } j \text{ is a dangling node,} \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

2) Damping Factor: We can see that in the graph depicted in Figure 3, Nodes **E**, **F**, **G** & **H** can make a subgraph with no outgoing links to the rest of the graph. Eveidently, all the probablities from rest of graph would flow into the subgraph

with increasing iterations and the other nodes of the graph would get a pagerank of 0 in the end.

This problem can solved with introducing a damping factor, i.e, there is some probality that the random surfer does not always follow outgoing links but may jump to a page randomly. Let the probablity of the surfer to follow any outgoing link be d. Then probablity of jumping to any random node is $1-d$. Then instead of matrix $\mathbf{A}$, we can define a new transition matrix $\mathbf{M}$ as given below:

$$\mathbf{M} = d\mathbf{A} + \frac{1-d}{N}\mathbf{J}$$

where $\mathbf{J}$ is a square matrix with all elements as 1 and d is called the **damping factor**

Damping factor d is generally taken to be 0.85.

Therfore, probablity can flow out out a subgraph of a reducible graph due to realistic modelling of user behaviour.

III. Implementation Details of PageRank

A. Non-Distributed

A python implementation of the above discussed PageRank algorithm is show below. The code is well comented and self-documenting.

```
import csv
import sys

import numpy as np
from numpy.testing import assert_allclose

damping = 0.85 # damping factor
max_iter = 10 # maximum iterations
tol = 1e-7 # maximum tolerance between
    pagerank results of two consequtive
    iterations


with open(sys.argv[1], "r") as f:
   reader = csv.reader(f, delimiter=" ")
   n = int(sys.argv[2])
   print(n)
   A = np.zeros((n, n), dtype=float)
   # src = source, dst = destination
   for src, dst in reader:
      # populate the A matrix in index (dst,
          src)
      A[int(dst) - 1][int(src) - 1] = 1.0
   for i in range(n):
      # handle dangling nodes
      if np.all(A[:, i] == 0.0):
         A[:, i] = 1.0 / n
      else:
         A[:, i] = A[:, i] / np.sum(A[:, i])

# create M matrix from A, incorporating
    damping factor
M = damping * A + ((1 - damping) / n) *
    np.ones((n, n), dtype=float)

# create initial pagerank
x = np.ones((n,), dtype=float) / n
x_trail = np.copy(x)

for _ in range(max_iter):
   print(x)
   x = np.inner(M, x_trail)
   if np.max(np.abs(x - x_trail)) <= tol:
      break
   x_trail = x


print(x)
```

B. MapReduce version of PageRank

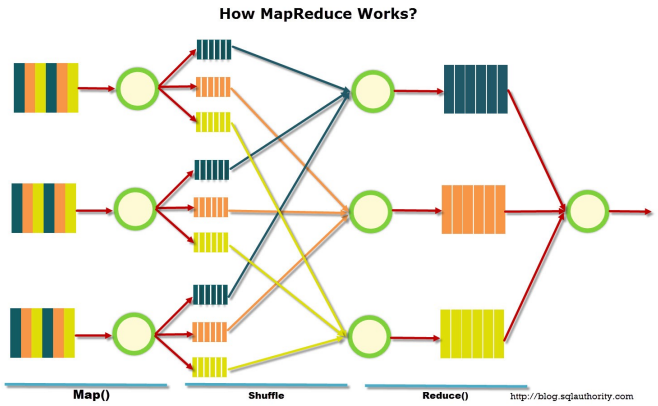

Fig. 4. A simple representation of MapReduce paradigm

The input file consists of several lines of two space seperated integers representing source and destination page numbers. An example is given below.

```
89 340
271 280
25 247
456 57
396 160
169 483
.
.
.
i j
```

The implementation consists of 3 pairs of mapper and reducer. We would discuss all 3 pairs in great detail.

1) Pair I: Mapper #1 takes each line of input, seperates the source and destination, subtracts 1 from both source and destination because the in the original input, indexing starts from 1. The output from the mapper is sorted lexicographically by Hadoop.

Reducer #1 converts the input into an adjacency list representation. It also takes care of listing the nodes which have no outgoing links. The output looks like the one shown below.

```
0 204
1 37 66
2 36 90 165 170 314 331
3 32 48 311 486
```

```
4 26 183 217 350
5
6 199 214 266 289 458 494
7 60 126 145 149 405 440
8 78
.
.
.
```

The first number in each line is the concerned node and others are the destination of outgoing links from that node.

2) Pair II: Before going into implementation of Mapper #2, we encounter a problem. The matrix to be constructed $\mathbf{M}$ be guaranteed to be dense, due to the incorporation of the damping factor d. Let us suppose that N is large, say 5000, then number of elements in matrix is N^2, i.e., 25000000, which is not great for matrix multiplication, and even sorting. However we can reconvert $\mathbf{M}$ into a sparse matrix using some minor modifications.

$$\mathbf{M} = d\mathbf{A} + \frac{1-d}{N}\mathbf{J}$$
$$\mathbf{x}' = \mathbf{M}\mathbf{x} = d\mathbf{A}\mathbf{x} + \frac{1-d}{N}\mathbf{J}\mathbf{x}$$
$$x'_i = dA_{ij}x_j + \frac{1-d}{N}J_{ik}x_k$$

as J_{ik} is 1 for every element

$$x'_i = dA_{ij}x_j + \frac{1-d}{N}\sum_k x_k$$
$$x'_i = dA_{ij}x_j + \frac{1-d}{N}$$

Einstein indicial notation is followed everywhere.

As $d\mathbf{A}$ is sparse, matrix-vector multiplication in the end becomes computationally inexpensive.

Mapper #2 takes the adjacency list and counts the number of outlinks for each page. Let the page in question be i and number of outlinks be L_i, then two cases arise.

- If L_i is 0, then N lines are appended with the format $(j, i, d * \frac{1}{N})$ where j runs from 0 to $N-1$.
- If L_i is not 0, then L_i lines are appended with the format $(j, i, d * \frac{1}{L_i})$ where j is destination of all outlinks from i.

Hadoop sorts the output lexicographically and pipes the output to reducer #2, which forwards the STDIN directly into STDOUT without any change.

3) Pair III: The last pair of mapper-reducer does the job of matrix-vector multiplication according to the following algorithm. The algorithm can be used only if the vector can be accomodated in the computer memory.

Algorithm 1 Calculate $\mathbf{v}' = \mathbf{M}\mathbf{v}$

Function Mapper $(< i, j, M_{ij} >, < k, v_k >)$
Store all the components of vector $\mathbf{v}$ *in memory*
for *all pairs of available* $i,\ j$ **do**
  emit $< i, M_{ij} * v_j >$
end for

Function Reducer $(< i, v_j >)$
for *all* v_j *belonging to the same* i **do**
  emit $< i, \sum v_j >$
end for

We can run this mapper-reducer pair for a specified number of times or to acheive a specified tolerance between two iterations.

C. Problems encountered during implementation

- Size of matrix $\mathbf{M}$ is typically huge, therefore, we need to find a way to make it sparse.
- Providing specific kind of input to the mapper so that it can be parallelized.
- For very large graphs in which pagerank vector can't be fit in the memory, new matrix-vector multiplication methods need to be explored.
- Currently, reducer can't be parallelized.

REFERENCES

[1] K. Shum (2013, April 3) Notes on PageRank Algorithm. Retreived from http://home.ie.cuhk.edu.hk/~wkshum/papers/pagerank.pdf
[2] Page, Lawrence & Brin, Sergey & Motwani, Rajeev & Winograd, Terry (1998). The PageRank Citation Ranking: Bringing Order to the Web.
[3] https://en.wikipedia.org/wiki/PageRank
[4] MapReduce Algorithm for Matrix Multiplication. Retreived from http://www.mathcs.emory.edu/~cheung/Courses/554/Syllabus/9-parallel/matrix-mult.html
[5] Dean, Jeffrey & Ghemawat, Sanjay. (2004). MapReduce: Simplified Data Processing on Large Clusters. Communications of the ACM. 51. 137-150. 10.1145/1327452.1327492.