Week 6 & 7 Lecture Summary

**Lecture**

In this week’s lecture, we discussed PageRank, both the linear algebra behind the algorithm and the algorithm itself. We started by examining an example graph

where the nodes denoted web pages and the links denoted hyperlinks. We used this example graph to understand how a link matrix is created and given the link matrix, how to calculate the normalized eigenvector or the rank for each web page. We also discussed many other linear algebra concepts as it relates to the PageRank algorithm including a modified link matrix, the Perron-Frobenius theorem and random walks. This lecture was very dense; even after watching the lecture, our group was still overwhelmed by the math since it had been a while since we took a Linear Algebra course. Nonetheless, we took what we could from the lecture. The readings helped us gain a better understanding of what was discussed during the lecture, however.

**Readings**

*The PageRank Citation Ranking: Bringing Order to the Web*

Lawrence Page, S. Brin, R. Motwani, T. Winograd

PageRank is an algorithm that takes into account the link structure of the Web to produce an importance score for each web page that can be used for information retrieval/search, user navigation, web traffic estimation, and many other tasks. The computation for PageRank considers the rank for each web page’s backlink and the sum of the number of forward links from each backlink-ed web page. The equation is recursive and must be iterated until convergence. PageRank in its simplest form is not capable of handling dangling links (i.e. links that point to a page with no outgoing links). To resolve this issue, dangling links are removed from the link index prior to computing the importance score for each web page and added back to the index once the PageRank computation is complete. The simplified version of PageRank intuitively corresponds to the probability distribution of a random walk on the graph of the web; since this is a stochastic process, the math behind PageRank is treated like so. The authors used PageRank in two search engines. The first was a simple titled-based search engine. Using this search engine, PageRank was able to serve important web pages given a user query in comparison to Altavista, a competing search engine during the late 1990s, which served seemingly random web pages given the same user query. The second search engine was a full text search engine called Google. Google was able to successfully accommodate the common case scenario where users provided underspecified queries with the intent to find commonly referenced web pages. The authors suggest several ways that the PageRank algorithm could be manipulated, optimized and further utilized including choosing good initial rank assignments to decrease the rate of convergence, adjusting the parameter matrix *E* to personalize page rankings, and using PageRank as a backlink predictor.

*The $25,000,000,000\* Eigenvector The Linear Algebra Behind Google*

Kurt Bryan and Tanya Leise

Google’s success in the search engine space stems from its PageRank algorithm. The algorithm aims to score the pages in a web to quantify a page’s importance and determine which pages to serve first in a search. The paper starts with a small web example consisting of four pages where the number of backlinks per page is used to calculate each page’s importance score. The first approach, though simple, doesn’t consider the page’s importance in the score; therefore, a modified approach is discussed where the score of page *j* is computed using the sum of all pages links to page *j.* To prevent pages from gaining extra influence by increasing the number of hyperlinks present in the page, we further modified the baseline approach by dividing the sum of page links to page *j* by the sum of outgoing links.

Though this final approach works well for computing importance scores, there were two cases where this approach causes issues. The first issue arises when dealing with webs that have non-unique rankings (i.e., when the dimension of the eigenspace exceeds one). To resolve this, a modified link matrix **M** is used where the original link matrix **A** and the column-stochastic matrix **S** where all entries are inverted is multiplied by a weighted average of both matrices and matrix **M** is used to calculate the importance scores instead of matrix **A.**

The second issue arises when dealing with webs with dangling nodes, pages with no outgoing links. To resolve this, the column-substochastic link matrix **A** would need to have an eigenvalue less than or equal to one and a Perron eigenvector (i.e. a corresponding eigenvector *x* with non-negative entries).

*Data Intensive Text Processing with MapReduce*, Chapter 5

Jimmy Lin and Chris Dyer

Graphs and graph algorithms are popular topics in computer science. Popular applications include graph search and path planning, clustering, MSTs, etc. Graph networks can be expressed as either adjacency matrices, which is generally preferred by mathematicians, or adjacency lists, which is generally preferred by computer scientists. The chapter covers two popular, well-studied graph problems. The first being the “single source shortest path problem”. Dijkstra’s algorithm is used to solve these type problems. Dijkstra’s algorithm, though optimized when using non-parallel computing, can not be implemented using the MapReduce framework since MapReduce doesn’t not support global data that is mutable and accessible by the mappers and reducers. Instead, to solve these type problems using MapReduce, a brute-force iterative program is suggested where the mapper computes all of the distances between the source and all nodes and the reducer selects the shortest path. The second topic covered was PageRank. When implementing PageRank using MapReduce, mappers are used to evenly divide up each node’s PageRank mass and reducers are used to sum the PageRank contributions at each destination node.

**Homework Assignments**

The homework assignment was split into two submissions. The first was just to answer the seventeen exercises from the eignenvector paper and the second was to implement the PageRank algorithm detailed in chapter five of *Data Intensive Text Processing with MapReduce* using python’s MRJob. Our group is still in the midst of completing both assignments. Both assignments are challenging given that the math and proofs required of us is intimidating and the iterative nature of PageRank. Nonetheless, we are making progress.

**Lab 5**

In this lab, we were introduced to GraphFrames, an Apache Spark package used to create graphs from DataFames. The package provides several niceties when operating with graphs such as calculating the in-degree of each node, motif finding, and the ability to query against the nodes and edges independently. Downloading the package and trying a few examples was straightforward since the GraphFrames documentation explained the steps for installation and provided examples for us to explore. The only “difficulty” we encountered was when trying the GraphFrames’ PageRank function using the example from the eigenvalue paper, we didn’t know how to extract the pagerank results from the DataFrame into a list so that we could normalize the results; however, after we were able to figure it out after googling how to do so.