IMPLEMENTING AN ONLINE VERSION OF HYPERLINK-INDUCED TOPIC SEARCH (HITS) ALGORITHM

A CS 297 Project Report

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Amith Kollam Chandranna

May 2010

ABSTRACT

IMPLEMENTING AN ONLINE VERSION OF HYPERLINK-INDUCED TOPIC SEARCH (HITS) ALGORITHM

by Amith Kollam Chandranna

In general, search engines perform the ranking of web pages in an offline mode, which is after the web pages have been retrieved and stored in the database. The existing HITS algorithm operates in an offline mode of page rank calculation. In this project, we will implement an online mode of page ranking for this algorithm. This will improve the overall performance of the Search Engine.

The report also describes the research conducted on other search algorithms. These include studying the working of PageRank, SALSA and OPIC algorithms. The research also focuses on methods to efficiently calculate eigenvalues, eigenvectors, and other methods related to matrix operations. The understanding of these is very essential in building a working model of a modified HITS algorithm. Implementation details of Nutch search engine are discussed. This understanding will be helpful in implementing the solution for CS 298. Journal articles related to ranking algorithms, specifically focusing HITS are also analyzed to gain more thorough understanding.

Table of Contents

List of Figures

1. Introduction to Project

Hyperlink-Induced Topic Search (HITS) algorithm is one of the page ranking algorithms used by search engines. This was developed by Jon Kleinberg, a Computer Science Professor at Cornell University. This algorithm calculates the ranking of web pages in an offline mode. In this project, we will implement an online page ranking for this algorithm

Search engines perform their operations in two phases. In the first phase, this algorithm performs a crawl to gather all the web pages and stores these crawled web pages in a database. The particular format of storing these web pages differs from one search engine to another. However, these are stored in a compressed format and are indexed for faster retrieval.

The next phase involves parsing the content of the stored web pages. This step is essential in order to determine the relative ranking for each page. Ranking the web pages is a highly complex process. Some of the factors that make this complex are: billions of web pages, intricate connections among these web pages, different formats, different languages, etc. Apart from these, different search technologies have their own pros and cons. This in turn complicates the functioning of a particular search engine. The HITS algorithm generates ranking for web pages after they have been crawled and stored in a local database. This is technically described as generating the ranking in an “offline” mode. In this project, we propose to implement the online rank calculation mode. Page crawl and ranking of pages can be done simultaneously, but of course we need to start with the page crawl first. This will certainly enhance the efficiency and reduce the overall execution time required in the search process.

The scope of the project is to implement an online version of the HITS-based web page ranking algorithm. Hence, the existing HITS algorithm will be modified to provide this new functionality.

To implement the modified version of HITS, it is essential to understand the working of the other search algorithms. In section 2, the working of the Online Page Importance Calculation Algorithm (OPIC) will be discussed. Section 3 describes the overview of the PageRank. It also provides details about its pseudo code. Section 4 describes the Stochastic Approach for Link-Structure Analysis (SALSA) algorithm. In section 5, the implementation details of Nutch search engine is provided. Finally, section 6 describes the Hyperlink-Induced Topic Search (HITS) algorithm, Eigenvalues and Eigenvectors, and the Power method. The conclusion provides the outcome of CS 297 and the proposed future work.

2. OPIC Algorithm Deliverable

This deliverable simulated the working of the Online Page Importance Calculation Algorithm (OPIC). This tool was developed using Javascript and uses 10 × 10 static nodes to depict the working of a static network. A connection of nodes is activated or deactivated by selecting 1 or 0 on the adjacency matrix. This deliverable helps to understand the efficient ways to implement data structures. This will be useful in implementing the final solution of the CS 298 project.

The basic working of the OPIC algorithm can be defined as:

Initially, some “cash” is distributed to each page and each page when it is crawled distributes its current cash equally to all pages it points to. Cash can be defined as the numerical value allotted to each page. The static nodes of the matrix represent the web pages. This is recorded in the history of the page. The importance of a page is then obtained from the “credit history” of the page.

The idea is that the flow of cash through a page is proportional to its importance.

At each step, an estimate of any page k's importance is (H[k] + C[k]) / (G + 1), where H[k] represents history, C[k] represents cash and G is the total cash accumulated.

The algorithm is executed over a decided number of iterations until the acceptable rate of convergence is achieved.

Below is the snapshot of the tool:

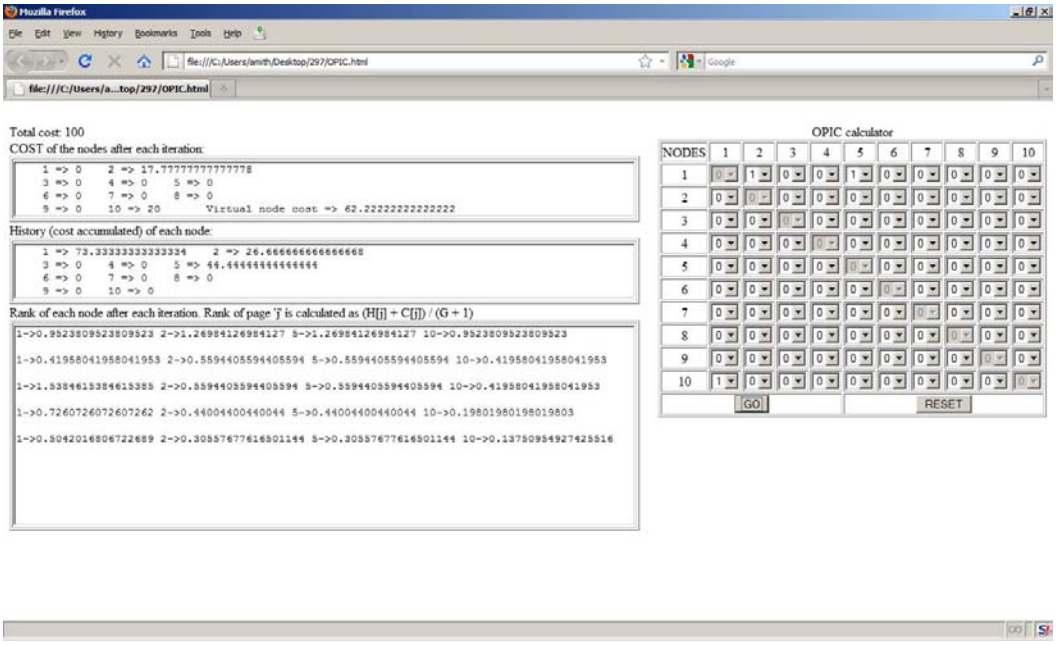

Figure 1: A simulation of OPIC algorithm
Source: SJSU CS Project portal. Retrieved May 4, 2010
http://www.cs.sjsu.edu/faculty/pollett/masters/Semesters/Spring10/amith/demo1.jpg

3. PageRank Algorithm Deliverable

In this scheme, the stored web pages are parsed to generate a ranking for each of them. Web pages on the internet have hyper-text links to other web pages and also other pages point to it. So, this algorithm iteratively calculates the importance of a given web page based on the recommendations from other web pages. Here, recommendations refer to the hyper-text links from other pages. Pages with higher recommendations have higher importance on the ranking scale.

The PageRank of a page P_i, denoted $r(P_i)$, is the sum of the PageRanks of all pages pointing into P_i: $r(P_i) = \sum r(P_i) / | P_j |$ where P_j is the set of pages pointing into P_i

It is convenient to perform computations in matrix form and hence this is represented as:

$\pi^{(k+1)T} = \pi^{(k)T} H$

where $\pi^{(k)T}$ and $\pi^{(k+1)T}$ are $1 \times n$ row vectors at k^{th} and $(k+1)^{th}$ iteration

H is the adjacency matrix of the graph.

Each iteration calculates this row vector over H, which is a sparse matrix. These iterations are performed until the desired rate of convergence is achieved. Then the normalized ranks of web pages are presented to the user. These ranks are in the range of 0 to 10.

Below image illustrates the pseudo code for this algorithm:

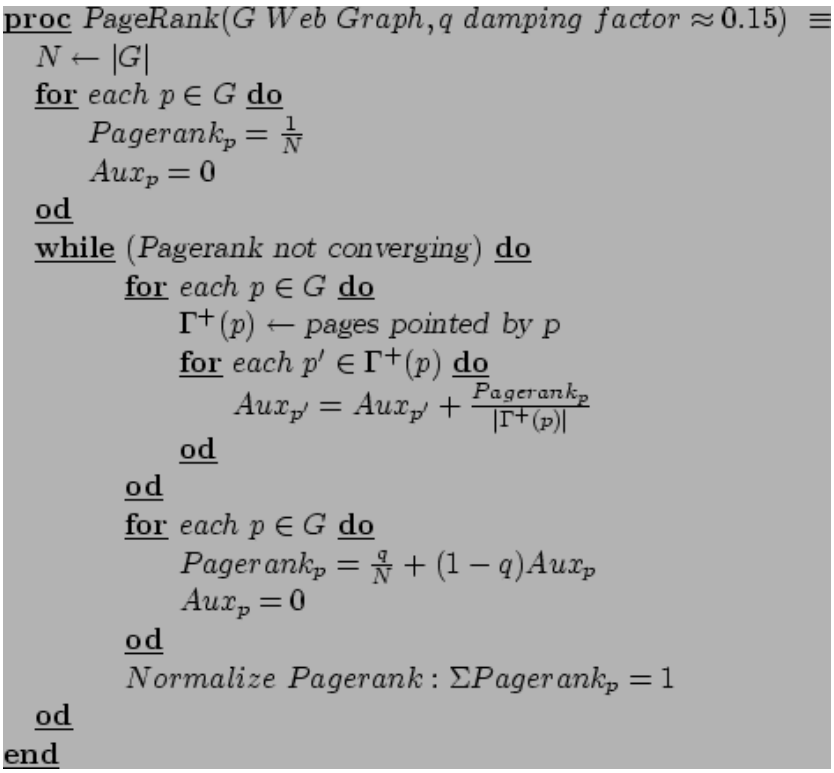

Figure 2: Pseudo Code of PageRank Algorithm
Source: Focused crawler, Retrieved April 25, 2010
http://combine.it.lth.se/CrawlSim/report/node20.html

To initialize the PageRank algorithm, we set $\pi^{(0)T} = 1 / n\, e^T$. The problem here was that the pages that accumulated more PageRank value at each iteration will monopolize the scores. To overcome this problem, the notion of a random surfer model was invented. This model describes the way in which a surfer navigates the web pages which have many outgoing hyperlinks. He then chooses one of the hyperlinks at random and continues this random decision process indefinitely. To model this activity, Brin and Page invented a new matrix called the Google matrix G. This matrix is described as

$$G = \alpha S + (1 - \alpha) 1/n\, ee^T$$

where α is a scalar between 0 and 1. This parameter controls the behavior of random surfer who follows the hyperlinks

So, the Google's adjusted PageRank method is

$$\pi^{(k+1)T} = \pi^{(k)T} G$$

This is the Power method applied to G. The Power method will be described in more details in later part of this report.

4. SALSA Algorithm Deliverable

This deliverable includes presentation slides describing the Stochastic Approach for Link-Structure Analysis (SALSA) algorithm and the working of the Nutch search engine. The understanding gains from this deliverable are essential in building the modified version of HITS algorithm, as the SALSA algorithm has many features similar to HITS. The presentation describes the advantages of SALSA and also highlights the various ways that it is more effective than PageRank algorithm.

SALSA is a variation of HITS algorithm (developed by Jon Kleinberg). It takes a result set R as input, and constructs a neighborhood graph from R in precisely the same way as HITS. Similarly, it computes an authority and a hub score for each vertex in the neighborhood graph, and these scores can be viewed as the principal eigenvectors of two matrices. However, instead of using the straight adjacency matrix that HITS uses, SALSA weighs the entries according to their in and out-degrees.

The approach is based upon the theory of Markov chains, and relies on the stochastic properties of random walks performed on our collection of pages. The input to our scheme consists of a collection of pages C which is built around a topic t.

Below is the formal definition of SALSA algorithm:

Let us build a bipartite undirected graph $G = (V_h, V_a, E)$ from our page collection and its link-structure:

$V_h = \{s_h \mid S \in C \text{ and out-degree}(s) > 0\}$ (the hub side of G).

$V_a = \{s_a \mid S \in C \text{ and in-degree}(s) > 0\}$ (the authority side of G).

$E = \{(s_h, r_a) \mid s \rightarrow r \text{ in } C\}$.

Each non-isolated page $s \in C$ is represented in G by one or both of the nodes s_h and s_a.

Each WWW link s -> r is represented by an undirected edge connecting s_h and r_a.

On this bipartite graph we will perform two distinct random walks. Each walk will only visit nodes from one of the two sides of the graph.

(Reference: “SALSA: The Stochastic Approach for Link-Structure Analysis”, 2001)

The Nutch is an open source search engine developed by the Apache organization. It was developed using Java and is easy to implement on any platform. The slides describe the basic functionalities provided by this search engine and its architecture.

5. Nutch Implementation Deliverable

In this deliverable, the actual implementation of Nutch was carried out on a Windows platform. This implementation gave a clear understanding of various issues faced in configuring a search engine. This helped to understand the integration of various components, tweaking the configuration settings and testing the results of the search engine. These ideas will help in implementing the solution for the final project.

5.1 Requirements for Installing Nutch

Java 1.4.x. Set NUTCH_JAVA_HOME to the root of your JVM installation. Apache's Tomcat 4.x and on Win32, install cygwin, for shell support.

After the installation of Nutch, we need to perform a crawl to acquire the web pages from the internet. There are two approaches to crawling: Intranet crawling, with the crawl command; and the whole-web crawling, with much greater control, using the lower level inject, generate, fetch and updatedb commands.

5.2 Intranet Crawling

Intranet crawling is more appropriate when you intend to crawl up to around one million pages on a handful of web servers.

5.3 Intranet: Configuration

To configure things for intranet crawling you must:

Create a flat file of root urls. For example, to crawl the Nutch site you might start with a file named urls containing just the Nutch home page. All other Nutch pages should be reachable from this page. The urls file would thus look like:

http://lucene.apache.org/nutch/

Edit the file conf/crawl-urlfilter.txt and replace MY.DOMAIN.NAME with the name of the domain you wish to crawl. For example, if you wished to limit the crawl to the apache.org domain, the line should read:

```
+^http://([a-z0-9]*\.)*apache.org/
```

This will include any url in the domain apache.org.

5.4 Intranet: Running the Crawl

Once things are configured, running the crawl is easy. Just use the crawl command. Its options include:

dir *dir* names the directory to put the crawl in.

depth *depth* indicates the link depth from the root page that should be crawled.

delay *delay* determines the number of seconds between accesses to each host.

threads *threads* determines the number of threads that will fetch in parallel.

For example, a typical call might be:

```
bin/nutch crawl urls -dir crawl.test -depth 3 >& crawl.log
```

Typically one starts testing one's configuration by crawling at low depths, and watching the output to check that desired pages are found. Once one is more confident of the configuration, then an appropriate depth for a full crawl is around 10.

Once the crawling is completed, one can skip to the Searching section below.

5.5 Searching

To search you need to put the nutch war file into your servlet container. (If instead of downloading a Nutch release you checked the sources out of SVN, then you will first need to build the war file, with the command ant war.)

Assuming you've unpacked Tomcat as ~/local/tomcat, then the Nutch war file may be installed with the commands:

rm -rf ~/local/tomcat/webapps/ROOT*

cp nutch*.war ~/local/tomcat/webapps/ROOT.war

The webapp finds its indexes in ./segments, relative to where you start Tomcat, so, if you've done intranet crawling, connect to your crawl directory, or, if you've done whole-web crawling, don't change directories, and give the command:

~/local/tomcat/bin/catalina.sh start

Then visit http://localhost:8080/ to perform the search.

(Reference: “Nutch Tutorial”, n.d)

6. HITS Algorithm Deliverable

This deliverable includes the implementation of a HITS simulation tool. This tool is built using Javascript. This accepts users input in the form a query string and adjacency matrix. This algorithm was developed by Professor Jon Kleinberg. The basic idea behind this algorithm is that all the web pages on the internet are categorized into two sets called Hubs and Authorities. Hubs define the web pages that have out going links to other important web pages and Authorities define the web pages that have incoming links from other important web pages.

Recursively, the algorithm iterates over these two sets to generate a hub and an authority value for each page. Depending on these values, the importance of web pages for a particular query is calculated and displayed to the user. The ranking module of HITS calculates the rank in an offline mode after the web pages have been downloaded and stored in the local database. In this project, we plan to implement the online version of this algorithm. For pseudo code of this

algorithm, please refer to “Google’s PageRank and Beyond” (“Google’s PageRank and Beyond”, 2006)

The pseudo code for HITS algorithm can be described as:

Initialize $y^{(0)} = e$. where e is the column vector of all ones.

Below is the iteration that is carried until convergence,

$x^{(k)} = L^T y^{(k-1)}$

$y^{(k)} = L x^{(k)}$

$k = k + 1$

Normalize $x^{(k)}$ and $y^{(k)}$

The equations in step 1 and step 2 can be simplified to

$x^{(k)} = L^T L x^{(k-1)}$

$y^{(k)} = L L^T y^{(k-1)}$

The above two equations define the iterative power method for computing the dominant eigenvector for the matrices L^TL and LL^T. L^TL is called the authority matrix and LL^T is known as hub matrix. Computing, the authority vector x and the hub vector y can be viewed as finding the dominant right-hand eigenvectors of L^TL and LL^T respectively.

6.1 Eigenvalues and Eigenvectors

For a given matrix A, the scalars λ and the vectors $x_{n\times 1} \neq 0$ satisfying $Ax = \lambda x$ are the respective eigenvalues and eigenvectors for A. The eigenvalues of $A_{n\times n}$ are the roots of the characteristic polynomial $p(\lambda) = \det(A - \lambda I)$, where det(*) denotes the determinant and I is the identity matrix. The degree of $p(\lambda)$ is n and A has n eigenvalues, but some may be complex numbers. The calculation of eigenvalues and eigenvectors is an essential part of algorithms like PageRank and HITS.

6.2 The Power Method

This method is used to calculate the Hub and Authority scores in the HITS algorithm. The standard method for finding the eigenvalues of a matrix A is to solve for the roots λ of $| A - \lambda I | = 0$. This however is totally impracticable when A is large. Even evaluating the determinant of an $n \times n$ matrix is a huge task when n is large, and on top of that we would have to solve the resulting n^{th} - degree polynomial equation for λ .

The power method provides a very straightforward way of computing one eigenvalue (usually the largest) λ, and corresponding eigenvector e, of a matrix. It is an iterative method, in which we start with an initial guess x_0 and generate a sequence of approximations x_k which converges as $k \to \infty$. Below is the pseudo code for this method:

Choose initial guess x_0 (x_0 is initial vector)

Do k = 1 to m.

Set $y_k = A_{xk-1}$.

Set α_k = the largest element of y_k (in absolute value).

Set $x_k = y_k/\alpha_k$.

Repeat from step 3.

(Reference: “The Power Method”, n.d)

7. Conclusion

In this project, we will implement an online ranking version of the HITS algorithm. Ranking algorithms are considered to be the core of any search engine. Several benefits can be obtained by implementing this modified algorithm. Primary benefits include efficient use of resources and also improves the overall performance of the search engine.

Currently, the project is still in the analysis phase and more information related to the specific implementation is to be researched and gathered. This includes creating small test models, researching related journal articles, analyzing reference guides, generating test cases, and comparing test results with other models. This will help to understand the overall processes involved in successfully implementing the modified HITS algorithm.

Based on this analysis, the solution for the final project (CS 298) will be implemented. The tools implemented to simulate OPIC and HITS will help in the understanding of data structures used as well as efficient ways to store and retrieve data

Future work will involve carrying on further research and analysis while using the findings from CS 297 to implement the final solution. This will involve using the Javascript HITS implementation as the basis for building the modified HITS online ranking algorithm. This will be tested on the live data derived from crawling the internet and comparing the results with other search algorithms. This evaluation will be helpful in comparing the effectiveness of the project.

References

N. Langville, Amy., & D. Meyer, Carl. (2006). *Google's PageRank and Beyond*. Princeton University Press.

Search Engine Architecture. (n.d.). Retrieved April 25, 2010 from IBM web site: http://www.ibm.com/developerworks/web/library/wa-lucene2/figure1.gif

Pseudo Code of PageRank Algorithm. (n.d.). Retrieved April 25, 2010 from Combine System web page: http://combine.it.lth.se/CrawlSim/report/node20.html

Nomura, Saeko., Toru Ishida, Satoshi Oyama., & Hayamizu, Tetsuo. (2004). *Analysis and Improvement of HITS Algorithm for Detecting Web Communities*. [Electronic version]. ACM Systems and Computers in Japan, Vol 35, Issue 13, 32 – 42.

Borodin, Allan., O. Roberts, Gareth., S. Rosenthal, Jeffrey., & Tsaparas, Panayiotis. (2005). *Link analysis ranking: algorithms, theory, and experiments*. [Electronic version]. ACM Transactions on Internet Technology (TOIT), Vol 5, Issue 1, 231 – 297.

Lempel, R., & Moran., S. (2001). SALSA: The Stochastic Approach for Link-Structure Analysis. [Electronic version]. ACM Transactions on Information Systems, Vol. 19, 131–160.

The Power Method. (n.d). Retrieved May 4, 2010 from University of Nottingham's web page: http://www.maths.nottingham.ac.uk/personal/sw/HG2NLA/pow.pdf

Nutch Tutorial. (n.d). Retrieved May 07, 2010 from Apache's web site: http://lucene.apache.org/nutch/tutorial.html