ACCELERATING RANKING SYSTEM USING WEBGRAPH

By

Padmaja Adipudi

A project submitted to the graduate faculty of The University of Colorado at Colorado Springs in partial Fulfillment of the Master of Science degree

DEPARTMENT OF COMPUTER SCIENCE

2007
This project for Master’s of Science degree by Padmaja Adipudi has been approved for the Department of Computer Science

By

______________________________
Dr. J. Kalita (Advisor)

______________________________
Dr. E. Chow (Committee Member)

______________________________
Dr. T. Chamillard (Committee Member)

______________________________
Date
ACKNOWLEDGEMENTS

Many people have shared their time and expertise to help me accomplish this project. First I would like to sincerely thank my advisor, Dr. Jugal K. Kalita for his guidance and help. And also many thanks to Dr. T. Chamillard and Dr. C. Edward Chow for their supports.

I wish to pay special tributes to the fellow engineers Srinivas Guntupalli, Sunil Bhave and Shawn Stoffer who provided constructive suggestions. I would like to thank Sonali Patankar who provided a large set of sample data.

Finally, I need to acknowledge that all the friends in the research team are of great help. Thank you!
# Table of Contents

1 Abstract............................................................................................................................................ 7

2 Introduction........................................................................................................................................ 8
   2.1 Cluster Rank Algorithm.................................................................................................................. 8
   2.2 Problem Statement.......................................................................................................................... 9
   2.3 Summary of Work .......................................................................................................................... 9
   2.4 Related Work ................................................................................................................................ 9

3 Implementation .................................................................................................................................. 12
   3.1 Workflow ........................................................................................................................................ 13
   3.2 Google’s PageRank ......................................................................................................................... 13
   3.3 Cluster Rank ................................................................................................................................... 14
   3.4 Source Rank ................................................................................................................................... 15
   3.5 Truncated PageRank ....................................................................................................................... 16
   3.6 Software & Packages Used ............................................................................................................ 17
   3.7 Database Implementation ................................................................................................................. 18
   3.8 Database Tables Added ................................................................................................................... 18
      3.8.1 Table Source ............................................................................................................................. 18
      3.8.2 Table PageRankBySource ....................................................................................................... 19
      3.8.3 Table PageRankTruncated ...................................................................................................... 20
      3.8.4 View vwURLLinkSource ........................................................................................................ 20
   3.9 Database Changed Made ................................................................................................................ 20
   3.10 Database Relationships ................................................................................................................ 22
   3.11 Module Implementation ................................................................................................................. 23
      3.11.1 Original Implementation .......................................................................................................... 23
      3.11.2 Current Implementation .......................................................................................................... 23
      3.11.3 Current Implementation – Module Details ............................................................................ 24

4 Experimental Results .......................................................................................................................... 28
1 Abstract

Search Engine is a tool to find the information on any topic in the Web. The basic components of a Search Engine are Web Crawler, Parser, Page-Rank System, Repository and a Front-End. In a nutshell here is how the Search Engine operates. The Web Crawler fetches the web pages from Web, the Parser takes all downloaded raw results, analyzes and eventually tries to make sense out of them. Finally the Page-Rank system finds the importance of pages, and the Search Engine lists the results in the order of relevance and importance.

In short, a Page-Rank is a “vote”, by all the other pages on the Web, about how important a page is. Studying the Web graph, which is used in Page-Rank System, is often difficult due to their large size. In Web graph the Web pages are represented as nodes and the hyperlinks between the Web pages are represented as directed links from one node to other node. Different kinds of algorithms were proposed because of the large Web graph to get efficient Page-Rank Systems.

The Needle is a Search Engine at UCCS for educational domains developed by a group of previous students at UCCS under the guidance of Dr. J. Kalita. The goal for this project is to accelerate the Page-Rank System of Needle Search Engine, at the same time upgrading the Search Engine with 1 Million URLs. The acceleration of the Page-Rank System will be accomplished by applying a package called “WebGraph” [1] with compression techniques to represent the Web graph compactly, and also compare the ranking efficiency, by using two recently published ranking algorithms called Truncated PageRank [7] and Source Rank [10]. Finally deploy the best to upgrade the Needle Search Engine with 1 Million pages.
2 Introduction

Search Engine technology was born almost at the same time as the World Wide Web [9]. The Web is potentially a terrific place to get information on almost any topic. Doing research without leaving your desk sounds like a great idea, but all too often you end up wasting precious time chasing down useless URLs if the search engine is not designed properly.

The dramatic growth of the World Wide Web is forcing modern search engines to be more efficient and research is being done to improve the existing technology. The design of the Search Engine is a tedious process because of the dynamic nature and sheer volume of the data.

Page-Rank system is a component of Search Engine to find the importance of a Web page relevant to search topic. PageRank [6] is a system of scoring nodes in a directed graph based on the stationary distribution of a random walk on the directed graph. A graduate student Yi Zhang implemented Cluster Rank algorithm [4], which is based on the famous Google’s PageRank algorithm [6]. In Google’s PageRank the importance of a page is based on the importance of parent web pages.

2.1 Cluster Rank Algorithm

- Group all pages in to clusters.
  - Perform first level clustering for dynamically generated pages
  - Perform second level clustering on virtual directory and graph density

- Calculate the rank for each cluster with the original PageRank [6] algorithm

- Distribute the rank number to its members by weighted average.

The Cluster Rank [4] is designed and implemented to achieve similar goals as that of existing PageRank [6] while providing the similar performance and providing an additional feature for managing similar pages in search results.
2.2 Problem Statement

The existing Page-Rank System of the Needle takes long update times. It took around 2 hours to calculate the Page Rank for 300,000 URL and it will take months to update the system with the World Wide Web because of sheer volume of data. A group of people from Italy developed a package called “WebGraph” [1] to represent the Web graph compactly, which resolves the long update times for the World Wide Web.

2.3 Summary of Work

The Page-Rank System of the Needle Search Engine is designed and implemented using Cluster Rank [4] algorithm, which is similar to famous Google’s PageRank [4] algorithm. Google’s PageRank [4] algorithm is based on the link structure of the graph. A “WebGraph” [1] package is used to represent the graph in most efficient manner, which helps in accelerating the ranking procedure of the World Wide Web. Two latest Page-Rank algorithms called Source Rank [10], Truncated PageRank [7] are taken to compare the existing ranking system, which is Cluster Rank [4], and deploy the best in the Needle Search Engine. Two attributes are taken in to consideration for selecting the best algorithm. The first one is the time and second one is human evaluation for the quality of the search. A survey is conducted with the help of the research team on finding the best algorithm on different search topics.

2.4 Related Work

The existing Page-Rank system of the Needle Search Engine takes long update time as the number of URLs increases. Research was done on the published ranking system papers, and below are the details of those papers.

There is a paper called “Efficient Computation of page-rank” written by Taher H. Haveliwala [3]. This paper discusses efficient techniques for computing Page-Rank, a ranking metric for hypertext documents and showed that the Page-Rank can be computed for very large sub graphs of the Web on machines with limited main memory. They
discussed several methods for analyzing the convergence of Page-Rank based on the induced ordering of pages.

The main advantage of the Google’s PageRank [6] measure is that it is independent of the query posed by user, this means that it can be pre computed and then used to optimize the layout of the inverted index structure accordingly. However, computing the Page-Rank requires implementing an iterative process on a massive graph corresponding to billions of Web pages and hyperlinks. There is a paper written by Yen-Yu Chen and Qingqiang gan [2] on Page-Rank calculation by using efficient techniques to perform iterative computation. They derived two algorithms for Page-Rank and compared those with two existing algorithms proposed by Havveliwa [3], and the results were impressive.

In this paper [6], the authors namely Lawrence Page, Sergey Brin, Motwani and Terry Winograd took advantage of the link structure of the Web to produce a global “importance” ranking of every Web page. This ranking, called PageRank [6], helps search engines and users quickly make sense of the vast heterogeneity of the World Wide Web.

This paper introduces a family of link-based ranking algorithms that propagate page importance through links [7]. In these algorithms there is a damping function that decreases with distance, so a direct link implies more endorsement than a link through a long path. PageRank [6] is the most widely known ranking function of this family. The main objective of the paper is to determine whether this family of ranking techniques has some interest per se, and how different choices for the damping function impact on rank quality and on convergence speed. The Page Rank is computed similar to Google’s PageRank [6], except that the supporters that are too close to a target node, do not contribute to wards it ranking. Spammers can afford spam up to few levels only. Using this technique, a group of pages that are linked together with the sole purpose of obtaining an undeservedly high score can be detected. The authors of this paper apply only link-based methods that are they study the topology of the Web graph with out looking at the content of the web pages.
In this paper [10], they develop a spam-resilient Page-Rank system that promotes a source-based view of the Web. One of the most salient features of the spam-resilient ranking algorithm is the concept of influence throttling. Through formal analysis and experimental evaluation, they show the effectiveness and robustness of our spam-resilient ranking model in comparison with Google’s PageRank [6] algorithm.

The need to run different kinds of algorithms over large Web graph motivates the research for compressed graph representations that permit accessing without decompressing them [1]. At this point there exists a few such compression proposals, some of them are very efficient in practice.

Studying the Web graph is often difficult due to their large size [1]. It currently contains some 3 billion nodes, and more than 50 billion arcs. Recently, several proposals have been published about various techniques that allow storing a Web graph in memory in a limited space, exploiting the inner redundancies of the Web. The WebGraph [1] framework is a suit of codes, algorithms and tools that aims at making it easy to manipulate large Web graphs. The WebGraph can compress the WebBase graph [12], (118 Mnodes, 1Glinks) in as little as 3.08 bits per link, and its transposed version in as little as 2.89 bits per link. It consists of a set of flat codes suitable for storing Web graphs (or, in general, integers with power-law distribution in a certain exponent range), compression algorithms that provide a high compression ratio, algorithms for accessing a compressed graph without actually decompressing it (decompression is delayed until it is actually necessary, and documentation and data sets.
3 Implementation

A package called “WebGraph” [1] is used to represent the graph compactly. This package is developed in Java. The existing Page-Rank system is developed using Perl. A Perl library called “Inline-Java” is used to call the java modules of WebGraph [1] package to reuse the existing Perl code of the Cluster Rank [1] algorithm. Here is simple work flow diagram.

Listed below is the snippet of code that shows how to call Java from a Perl module:

```perl
Use Inline java => <<’DATA’;

/** JAVA Code Begins **/
import it.unimi.dsi.webgraph.ImmutableGraph;
...
public class MyGraph extends ImmutableGraph{
...
ImmutableGraph graph;
public MyGraph(String basename) throws IOException{
    graph = ImmutableGraph.load( basename );
}
...
public int getSuccCount( int n ) throws IOException ...{
    ...
}
}/** JAVA Code Ends **/

DATA

## Perl Code Begins ##
my $vargraph, $varnode, varcount;
$vargraph = new MyGraph("bvnodein");
$Id = 2;
$varcount = $vargraph->getSuccCount($Id);

The Page-Rank system gets the information stored by Crawler. WebGraph [1] package generates the compressed graph by taking a graph that is in ASCII graph format. In ASCII graph the first line contains the number of nodes ‘n’, then ‘n’ lines follow, the i-th
line containing the successors of node ‘i’ in increasing order (nodes are numbered from 0 to n-1). Successors are separated by a single space. This compressed graph is given as input to the Page-Rank system for calculation.

3.1 Workflow

![Workflow Diagram]

3.2 Google’s PageRank

The three algorithms Cluster Rank [4], Source Rank [10], Truncated PageRank [7] are based on the famous Google’s PageRank.

The published Page Rank algorithm can be described in a very simple manner:

\[
PR(A) = (1-d) + d \left( \frac{PR(T_1)}{C(T_1)} + \ldots + \frac{PR(T_n)}{C(T_n)} \right)
\]
In the equation above:

\[ \text{PR(Tn): Each page has a notion of its own self-importance. That’s “PR(T1)” for the first page in the web all the way up to PR(Tn) for the last page.} \]

\[ \text{C(Tn): Each page spreads its vote out evenly amongst all of its outgoing links. The count, or number, of outgoing links for page 1 is C(T1), C(Tn) for page n, and so on for all pages.} \]

\[ \text{PR(Tn)/C(Tn): if a page (page A) has a back link from page N, the share of the vote page A gets is PR(Tn)/C(Tn).} \]

\[ \text{d: All these fractions of votes are added together but, to stop the other pages having too much influence, this total vote is "damped down" by multiplying it by 0.85 (the factor d).} \]

The definition of \( d \) also came from an intuitive basis in random walks on graphs. The idea is that a random surfer keeps clicking on successive links at random, but the surfer periodically “gets bored” and jumps to a random page. The probability that the surfer gets bored is the dampening factor.

\[ (1 - d): \text{The }(1 - d)\text{ bit at the beginning is a probability math magic so the "sum of all Web pages" Page Rank is 1, achieved by adding the part lost by the d(....) calculation. It also means that if a page has no links to it, it still gets a small PR of 0.15 (i.e. 1 – 0.85).} \]

### 3.3 Cluster Rank

The original PageRank algorithm is applied on Clusters and then the rank is distributed to the members by weighted average.

1. Group all pages into clusters.

Perform first level clustering for dynamically generated pages

Perform second level clustering on virtual directory and graph density

2. Calculate the rank for each cluster with the original PageRank algorithm.
3. Distribute the rank number to its members by weighted average by using

\[ PR = CR \times \frac{Pi}{Ci}. \]

The notations here are:

PR: The rank of a member page
CR: The cluster rank from previous stage
Pi: The incoming links of this page
Ci: Total incoming links of this cluster.

### 3.4 Source Rank

The original PageRank algorithm is applied on Sources and then the rank is distributed directly to the members.

1. Group all pages into Sources based on “Domain”.
2. Calculate the rank for each Source with the original PageRank algorithm.
3. Distribute the rank number to its members by weighted average by using

\[ PR = SR \times Si. \]

The notations here are:

PR: The rank of a member page
SR: The source rank from previous stage
Si: Total incoming unique links of this source
3.5 **Truncated PageRank**

Truncated Page rank is link based ranking function that decreases the importance of neighbors that are topologically close to the target node. A damping function is introduced to remove the direct contribution of the first level of the linking.

They suggest generalization to the PageRank equation to:

The rank propagates through links.

We can calculate the Page Rank of a page by summing up contributions from different distances.

\[
PR(p) = \sum \alpha t \cdot Mt = \sum \text{damping}(t) \cdot Mt
\]

The notations here are:

C: Normalization constant

\(\alpha\): The damping factor
3.6 Software & Packages Used

**WebGraph:**

WebGraph is a java package used to represent the graph compactly.

**Java:**

Java is used as the programming language to use the WebGraph package.

**jdbc::mysql:**

To update the MySQL database tables with the Page Rank information in the Truncated PageRank module written in Java.

**Inline-java:**

This is a Perl library to access the java modules from Perl.

**Perl:**

Perl (version 5.8.8) is used as the programming language. A fast interpreter, its features to handle and manipulate strings and relatively small memory signatures of its modules make it an ideal language for this project.

**MySQL:**

The database is designed and implemented using MySQL v3.23.58 and v4.1.1. MySQL is free, scalable and Perl has a rich API for MySQL.

**PhpMyAdmin:**

It is a good and intuitive front-end user interface for the MySQL database. Many features are provided to create, manipulate and manage databases and users in the MySQL environment. One can also see and adjust MySQL environment variables.

**Apache:**
Apache server v2.0.54 is used for the machines to communicate using the CGI module.

**Software used for documentation:**

Microsoft Excel was used for the diagrams and Microsoft Word was used to write the project report.

### 3.7 Database Implementation

The original Needle [4] database consists of 15 tables and 2 views. Changes were made to this Needle database schema in order to accommodate the Source Rank [10] and Truncated PageRank [7] calculations.

Three new tables namely Source, PageRankBySource, PageRankTruncated and view named vwURLLinkSource are added. An explanation of each table and changes made to the database schema is shown below.

### 3.8 Database Tables Added

#### 3.8.1 Table Source

This table was added to store the source ids, which are required to compute the weighted Page Rank for the URLs using the Source Rank algorithm. Listed below are the table columns and their purpose.

- **source_id:** To compute the Page Rank of URLs using the Source Rank algorithm, the URLs are grouped into individual sources based on domain name of the URL. This column uniquely identifies that source.

- **base_url:** The base URL of the source

- **source_rank:** The rank given to the source which will be used later during the source rank computation.
o source_rank_date: The date on which the source was computed

o out_link_count: The number of out links of the source

o in_link_count: The number of in links of the source

o cal_current_iter: The iteration number used to converge the source rank

o old_date1: The previous date 1 on which the Page Rank was computed

o old_prc1: The previous weighted Page Rank 1 of the URL

o old_date1: The previous date 2 on which the Page Rank was computed

o old_prc1: The previous weighted Page Rank 2 of the URL

3.8.2 Table PageRankBySource

This table was added to store the weighted Page Rank for the URLs computed using the Source Rank algorithm. Listed below are the table columns and the purpose.

o url_id: The URL id for which the weighted Page Rank is given. This is the primary key of the table.

o c_date: The date on which the weighted Page Rank was computed

o c_prc: The weighted Page Rank

o old_date1: The previous date 1 on which the Page Rank was computed

o old_prc1: The previous weighted Page Rank 1 of the URL

o old_date1: The previous date 2 on which the Page Rank was computed

o old_prc1: The previous weighted Page Rank 2 of the URL
3.8.3 Table PageRankTruncated

This table was added to store the weighted Page Rank for the URLs computed using the Truncated PageRank algorithm. Listed below are the table columns and their purpose.

- **url_id**: The URL id for which the weighted Page Rank is given. This is the primary key of the table.
- **c_date**: The date on which the weighted Page Rank was computed
- **c_prc**: The weighted Page Rank
- **old_date1**: The previous date 1 on which the Page Rank was computed
- **old_prc1**: The previous weighted Page Rank 1 of the URL
- **old_date2**: The previous date 2 on which the Page Rank was computed
- **old_prc2**: The previous weighted Page Rank 2 of the URL

3.8.4 View vwURLLinkSource

This view uses provides the information from URL and URLLinkStructure to obtain the out-links of each source. The view also makes sure that the url_id exists in both URL and URLLinkStructure tables and is not NULL. Listed below are the table columns and their purpose.

- **fromsource**: The source_id of URL table that acts as the from source id
- **tosource**: The source_id of URL table that acts as the to source id (out-link)

3.9 Database Changed Made

It has been noticed that there is scope to improve the performance of the SQL query execution in the existing Page-Rank System implementation. Changes were also made to the existing database schema in order to accommodate the Page-Rank systems using the
new algorithms namely Source Rank and Truncated Page Rank. Listed below are the details of the changes made.

- Index has been created for base_url of Cluster table to improve performance of the SQL query execution in the Perl modules.

- The PageRank table was populated using the PageRank module during the original Cluster Rank implementation, and used later in order to obtain the out-links of a specific URL. We replaced the need of this table by creating Web graph for the node out-links.

- Index has been created to sec_base_url of SecondLvlClusterWork table to improve the performance of SQL query execution in Perl modules.

- The column source_id has been added to URL table and index was created for source_id to accommodate the Source Rank computation.
3.10 Database Relationships
3.11 Module Implementation

3.11.1 Original Implementation

The original Needle Search Engine was implemented using the Perl programming language and MySQL as the backend database. After crawling the web pages, the web page URLs were stored in a MySQL database table called URL. The link structure between the URLs was stored in a table called URLLinkStructure.

This implementation generates the Page Rank for the URLs in a reasonable amount of time, as long as the URL linking structure (URL graph) is of small size. As the number of URLs grow with bigger URL linking structure, this ranking system becomes less efficient, in other words, this ranking system will take long update times and requires more machine resources such as memory and CPU to compute the Page Rank.

3.11.2 Current Implementation

In order to accelerate the original implementation of the ranking system, a package called WebGraph [1] is used in the current implementation. The WebGraph [1] package was developed by a group of people from Italy using Java programming language. By using this package, the URL linking structure, also called as ‘web graph’ can be represented compactly using compression techniques. The package provides several Java methods to access the compressed format of the web graph.

To make use of the WebGraph [1] package in the current implementation, a Perl library called Inline::Java is used. Using this library, the Perl programs can call the Java methods of the WebGraph [1] package.

To improve the SQL query performance in the original implementation, indexes were added on as needed basis to the backend database tables.

Two other recent ranking algorithms namely SourceRank [10] and the Truncated PageRank [7] were implemented to compare against the original Cluster Rank algorithm [4], in terms of the efficiency and quality of the ranking system. In order to accommodate
these two algorithms in the current implementation, the original database schema has been changed as needed. The three algorithms will be compared for efficiency, using the metrics generated against URLs of sizes 300K, 600K and 4 Millions. The quality will be measured by conducting a survey among a group of people who will perform the keyword queries on three different search engines that implement these algorithms.

3.11.3 Current Implementation – Module Details

The implementation has 3 phases namely graph generation, rank generation and search engine. The modules listed below explain these in detail.

3.11.3.1 Graph Generation Modules

To make use of the WebGraph package, the URL link structure is represented in ASCII format, in the form of a file named basename.graph-txt. The first line contains the number of nodes ‘n’, then ‘n’ lines follow the i-th line containing the successors of the node ‘i’ in the increasing order (nodes are numbered from 0 to n-1). The successors are separated by a single space.

The ASCII formatted URL link structure will then be converted in to the compressed format called BVGraph format (Boldi-Vigna Graph format, in the name of WebGraph package authors Paolo Boldi & Sebastiano Vigna). The compressed BVGraph is described by a graph file (with extension .graph), an offset file (with extension .offsets) and a property file (with extension .properties)[13].

The BVGraph can be generated from an ASCII formatted Graph using the command listed below:

```
java it.unimi.dsi.webgraph.BVGraph -g ASCIIGraph example bvexample
```

where example is the basename of the ASCII formatted graph (example.graph-txt) and the bvexample if the name for the resulting BVGraph (bvexample.graph, bvexample.offsets, bvexample.properties).
In the current implementation, we represent the URL link structure (graph) in two different ASCII format files named nodein.graph-txt and nodeout.graph-txt. The nodein.graph-txt represents each node and its in-links. The nodeout.graph-txt represents the node and its out-links.

The ASCII formatted URL link structure will then be converted into the compressed BVGraph format. The BVGraph will be represented as bvnodein.graph, bvnodein.offsets, and bvnodein.properties for the ASCII graph nodein.graph-txt. The BVGraph will be represented as bvnodeout.graph, bvnodeout.offsets, and bvnodeout.properties for the ASCII graph nodein.graph-txt.

a) nodegraphin.pl: This Perl module will generate the ASCII formatted nodein.graph-txt by executing SQL queries against URLLinkStructure table. Once the nodein.graph-txt is available, the same module will generate the compressed BVGraph files named bvnodein.graph, bvnodein.offsets and bvnodein.properties.

b) nodegraphout.pl: This Perl module will generate the ASCII formatted nodeout.graph-txt by executing SQL queries against URLLinkStructure table. Once the nodeout.graph-txt is available, the same module will generate the compressed BVGraph files named bvnodeout.graph, bvnodeout.offsets and bvnodeout.properties.

The Source Rank algorithm groups up a set of nodes into sources. During the Source Rank calculation, the algorithm will require to find the in-links for each source. In order to accelerate the Source Ranking system, the Source linking structure is represented in BVGraph format.

a) sourcegraphin.pl: This Perl module will generate the ASCII formatted sourcein.graph-txt by executing SQL queries against vwURLLinkSource view. Once the sourcein.graph-txt is available, the same module will generate the compressed BVGraph files named bvsourcin.graph, bvsourcin.offsets and bvsourcin.properties.
3.11.3.2 Rank Generation Modules

- **ClusterRank**: The Cluster Ranking system uses two Perl modules namely clustering.pl and clusterrank.pl as described below.

  a) clustering.pl: In this module, two phases occur namely the first level clustering and the second level clustering. The first level clustering selects each url id from the URL table, based on the content of the URL, finds out if it belongs to an existing cluster. If it belongs to an existing cluster in Cluster table, updates the cluster id in URL table. If it does not, creates a new cluster in Cluster table, updates the cluster id in URL table. The second level clustering will calculate the density for each cluster in Cluster table, approves it depending upon the density threshold value.

  b) clusterrank.pl: In this module, we generate the cluster rank for each cluster based on Google’s PageRank algorithm. The Page Rank for each URL contained in the Cluster will then be calculated and stored in a table named PageRankByCluster.

- **SourceRank**: The Source Ranking system uses two Perl modules namely sourcing.pl and sourcerank.pl as described below.

  a) sourcing.pl: In this module, the sourcing selects each url id from the URL table, based on the domain of the URL, finds out if it belongs to an existing source. If it belongs to an existing source in Source table, updates the source id in URL table. If it does not, creates a new source in Source table, updates the source id in URL table.

  a) sourcerank.pl: In this module, we generate the source rank for each source based on Google’s PageRank algorithm. The Page Rank for each URL contained in the Source will then be calculated and stored in a table named PageRankBySource.

- **Truncated PageRank**: The Truncated PageRank Ranking system uses a Java module namely TruncatedPageRank.java as described below.
a) **TruncatedPageRank.java:** In this module, the truncation eliminates the neighbors of the URL based on the predefined truncation factor. The rank for the remaining URLs will be calculated and stored in a table named PageRankTruncated.

### 3.11.3.3 Search Engine Modules

- **Search Engine:** The Search Engine gets the query words from users, it then selects the URLs, the keyword total_weight from Keyword table and the Page-Rank from the Page Rank table associated to the Algorithm used in the Search Engine. For example if Cluster rank algorithm is used the Page Rank table will be PageRankByCluster. The keyword weight is multiplied by the Page Rank value for each URL. The URLs then be displayed to the user in the descending sort order by the multiplied value.

  a) **search.pl:** This module presents the Search Engine UI to the end user. Three different forms, one for each algorithm namely Cluster Rank, Source Rank and Truncated Page-Rank will be listed. The User will be able to perform the search using any of the three implemented algorithms using this UI.

  b) **search_list.pl:** This module gets the query keywords and displays search results using the PageRankByCluster table

  c) **search_source.pl:** This module gets the query keywords and displays search results using the PageRankBySource table

  d) **search_truncated.pl:** This module gets the query keywords and displays search results using the PageRankTruncated table

  e) **search_sublist.pl:** This module will group the similar pages during the display of search results.
4 Experimental Results

4.1 Experimental Data Setup

The experiments were conducted on 3 different data sets with URLs of 300K, 600K and 4 Million. Importing the data from the Crawler was tedious and time consuming because of the large size of the data. The Crawler, Ranking system, Parser modules uses different database schemas. To setup the data for the current ranking system for the above listed data sets, the following steps were taken:

1. Identified the URL and URLLinkStructure tables generated by the Crawler module with sufficient number of URLs such as 600K and 4 Million. The database with these tables was on a server machine (128.198.144.19) that is different than the server machine (128.198.144.16) where the current project has been implemented.

2. Created/Copied necessary tables needed by the search engine from the search engine server machine to this database. These tables are namely Crawler, Dictionary, KeyWord, KeyWordWork, TextParser and PageLocation. Executed the Parser in order to populate these new tables using

- The URL and URLLinkStructure tables
- Other data files such as crawled documents generated by Crawler module.

3. Took a backup of these database tables needed by the Page-Rank system. Transported the backup file from the Crawler machine to Search Engine machine.

4. Restored the tables on to the database on the Search Engine machine. Made necessary database schema changes such as:

- Adding new columns to accommodate the Page-Rank calculations using algorithms namely Source Rank and Truncated PageRank
- Adding indexes to the tables obtained from the Crawler machine to improve the performance of SQL queries during the Page-Rank generation. The indexes were added to columns of the tables that were used in the WHERE clause of the SQL query coded in the Perl and Java modules.

4.2 Challenges & Key Observations

The original implementation uses the multiple database tables to compute the Page Rank. The linking structure between the URLs, which is the key factor while computing the Page Rank, was also represented in the form a database called URLLinkStructure. Most of the Page-Rank algorithms use the in-links and out-links of URL while computing the Page Rank of that URL.

As the number of URLs and the linking structure grow, it becomes complex and time consuming, to obtain the linkage information by sending repeated SQL queries against the huge URLLinkStructure table. It helps if the in-link and the out-link information is readily available during the Page Rank computation. A package called WebGraph was developed by a group of people from Italy to achieve this purpose. When the URL link structure is represented in an ASCII text file (ASCII web graph) in certain format, using the Classes and Methods described in this JAVA based package, the in-link, out-link information can be accessed very efficiently, as a result of which the Page-Rank computation times can be reduced significantly.

In brief, in ASCII web graph, each node’s in-links or out-links are listed in a single line separated by space in text file starting node 0 from second line to node n where n is the total number of nodes which is listed in the first line. Using the WebGraph package, this ASCII Graph can then be compressed and an equivalent BVGraph of significantly smaller in size compared to original ASCII graph can be generated. During the Page Rank computation, using the methods provided by the WebGraph package, the BVGraph of in-links can be loaded, the in-links of a node (successors) and the total number of in-links of a node can be efficiently accessed during the Page-Rank computation. The same is true for BVGraph of out-links.
It has been observed that the BVGraph can be loaded mainly in two different ways before they can be accessed, namely load and loadOffline methods. The load method will load the graph into memory. This helps to access the successors and an outdegree of a particular node directly by sending the node number as parameter to the methods provide by WebGraph package. This is very efficient but works only with smaller BVGraphs (approximately less than a million nodes). In case of large graphs, the graph needs to be loaded using loadOffline method. This method does not load the graph into memory. In order to get the successors and outdegree of a node n, we need to start from the beginning of the node 0, iterate the graph until we reach the node n and read the necessary information.

It has been observed that the total time it takes i) to generate the ASCII web graph using an optimized SQL query logic ii) to generate the equivalent BVGraph and iii) to access the node successor information using the graph, is lot less compared to the time it takes to access the same using SQL queries especially for large web graphs.

The current implementation does use the URL table during the Page-Rank computation. It has been observed that, instead of accessing the in-link and out-link information from the web graph, it will be very efficient if the number of in-links and out-links information is available for each URL in the URL table itself. Because of this, the current implementation performs the step of updating the in-link and out-link information in URL table for each URL, at the beginning of the Page-Rank computation process. It was proven that using this step, the overall time for Page-Rank, is reduced to a greater extent especially for large graphs.

The original implementation was written fully in Perl. Since the WebGraph package was developed in Java, in order to take advantage of the better features of the two languages, a Perl library called Inline::Java was used. During the page-rank computation of Source Rank and Cluster Rank, it has been noticed that separating out the BVGraph access to Java, Page Rank computation to Perl and passing the data between the two using Inline::Java concepts, was efficient.
For Truncated PageRank algorithm source code is available under GNU license is taken and made necessary changes to fit in to the current implementation. Java and the jdbc::mysql interface was used in order to store the Page Rank in to database table for later access during the search.

4.3 Future Upgrade of The Search Engine

The entire process of generating page-rank using algorithm namely Cluster Rank, Source Rank and Truncated PageRank was automated. The current implementation uses the URL table that contains more than 4 Million URLs.

Listed below are the steps to compute the Page-Rank for a different set of URLs.

- Make sure that the table URL provides the web page URLs and the table URLLinkStructure provides the link structure between the URLs respectively.
- If these two tables are available from a Crawler process, make sure that the tables namely Crawler, Dictionary, KeyWord, KeyWordWork, TextParser and PageLocation and made available in the same database
- Run the Parser module (perl textparser.pl) to update the KeyWord info.
- Copy all these tables to the database where the other tables listed in Appendix D.
- Make sure that the URL table has the columns as listed in Create URL table script of Appendix C (section 8.1). Make sure that the indexes are available on tables as documented in Appendix C (section 8.1).
- Run the Perl file called kickoff.pl to perform the steps listed below.
  - system("perl clustering.pl");
  - system("perl sourcing.pl");
  - system("perl nodegraphin.pl");
4.4 Time Comparisons For Cluster Rank Before & After Using WebGraph

The Original Search Engine used the Cluster Rank ranking system. This ranking system takes 6900 seconds, which is 1 hour 55 min per iteration for 289,503 nodes. In this algorithm we assume that it converges after 40 iterations.

Even though the ranking system calculates the rank for 289,503 nodes (URLs), all these pages are not considered during search results display process. This is because, not all these URLs are crawled by the Crawler module. For example, out of these 289,503 nodes, the Crawler module in the original Search Engine, crawled 54,201 pages. The Parser module of the Search Engine, works only on these crawled URLs to generate the keywords that will be used during the search results display.

The original Cluster Rank algorithm was run against Cluster Ranking system using WebGraph package. The ranking system takes 6780 seconds, which is 1 hour 53 min per iteration for 289,503 nodes. This experiment concludes that the ranking system takes less time for Page-Rank computation using the WebGraph package.

Listed below is the time comparison table for Cluster Rank before and after applying the WebGraph package.

<table>
<thead>
<tr>
<th>Function</th>
<th>Time for 300K URLs before using WebGraph (in seconds)</th>
<th>Time for 300K URLs after using WebGraph (in seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>system(&quot;perl nodegraphout.pl&quot;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>system(&quot;perl sourcegraphin.pl&quot;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>system(&quot;perl sourcerank.pl&quot;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>system(&quot;java -Xmx1000M TruncatedPageRank bnodeout -t 2&quot;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>system(&quot;perl clusterrank.pl&quot;)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table: Time per iteration for 300K URLs before/after using WebGraph

The overall time gain with WebGraph in Page-Rank computation using Cluster Rank algorithm is approximately 20% for 300K URLs. Listed below is the brief description for the overall time again using the WebGraph.

In prepareCR step above, we replaced the need of using the ‘PageRank’ table (and there by replacing the need of generating the ‘PageRank’ table solely for this purpose) by using the Web graph of out-links, for calculating the out-link information of URL. There appears to be a loss in terms of time it takes for this step, but in fact we did not have to generate the PageRank table which in directly saves significant amount of time.

In prepareIncomingCount, we replaced the need of updating the in-links of URLs using SQL query against URLLinkStructure, by using the Web graph of in-links. We gained significant amount of time in this step.

In doClusterRank step, we should have used the graph for Cluster in-links. But the original implementation of the Cluster Rank, generates a table during the ‘Clustering’ phase that is used for Cluster in-links. It has been observed that usage of the graph for Cluster in-links, in this step, does not gain much time compared to the usage of the table generated by Clustering phase (the first phase of original Cluster Rank implementation). This statement applies to graphs ranging from 300K URLs to 4 Million URLs based on

<table>
<thead>
<tr>
<th></th>
<th>Time before (seconds)</th>
<th>Time after (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>prepareCR</td>
<td>264</td>
<td>809</td>
</tr>
<tr>
<td>prepareIncomingCount</td>
<td>2288</td>
<td>192</td>
</tr>
<tr>
<td>doClusterRank (per iteration)</td>
<td>6900</td>
<td>6780</td>
</tr>
<tr>
<td>Total:</td>
<td>9452</td>
<td>7781</td>
</tr>
</tbody>
</table>
the experiments. The process of generating the Cluster in-links is made available for future development to leverage the graph’s usage over the table usage while dealing with huge sets of URLs over 4 Million, to improve the efficiency of the original implementation of the Cluster Rank algorithm.

Listed below is the time distribution represented in the form of Pie Chart for Cluster Rank before and after applying the WebGraph package.

Chart: Time distribution per iteration for 300K URLs before using WebGraph

Chart: Time distribution per iteration for 300K URLs after using WebGraph
Listed below is the time gain represented in the form of Bar Chart for Cluster Rank before and after applying the WebGraph package.

**Chart: Time gain per iteration for 300K URLs using WebGraph**

4.5 **Time Measure Between Algorithms Using WebGraph**

There are two recent algorithms called Source Rank, Truncated PageRank algorithm. The authors of these algorithms used the WebGraph package to represent the graph compactly during their Page-Rank calculation. Because of this reason, these two algorithms were considered for Page-Rank comparisons using WebGraph package. The experiments were performed using 300K, 600K and 4 Million URLs for Page-Rank calculation.

The time comparison of Page-Rank computation, for different sets of data using the three algorithms is listed below.

4.5.1 **For 300,000 URLs**

Listed below is the table that represents the time taken by each algorithm for the Page-Rank computation of 300K URLs.
<table>
<thead>
<tr>
<th>Step</th>
<th>Cluster Rank (in seconds)</th>
<th>Source Rank (in seconds)</th>
<th>Truncated PageRank (in seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node-In BVGraph</td>
<td>589</td>
<td>589</td>
<td>N/A</td>
</tr>
<tr>
<td>Node-Out BVGraph</td>
<td>857</td>
<td>857</td>
<td>857</td>
</tr>
<tr>
<td>Source-In BVGraph</td>
<td>N/A</td>
<td>92</td>
<td>N/A</td>
</tr>
<tr>
<td>Rank calculation</td>
<td>6780</td>
<td>660</td>
<td>22</td>
</tr>
<tr>
<td>(per iteration)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4.5.1: Time measure for Page-Rank computation between three algorithms (300 K)

**4.5.2 For 600,000 URLs**

Listed below is the table that represents the time taken by each algorithm for the Page-Rank computation of 600K URLs

<table>
<thead>
<tr>
<th>Step</th>
<th>Cluster Rank (in seconds)</th>
<th>Source Rank (in seconds)</th>
<th>Truncated PageRank (in seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node-In BVGraph</td>
<td>79</td>
<td>79</td>
<td>N/A</td>
</tr>
<tr>
<td>Node-Out BVGraph</td>
<td>68</td>
<td>68</td>
<td>68</td>
</tr>
<tr>
<td>Source-In BVGraph</td>
<td>N/A</td>
<td>3</td>
<td>N/A</td>
</tr>
<tr>
<td>Rank calculation</td>
<td>422</td>
<td>3</td>
<td>32</td>
</tr>
<tr>
<td>(per iteration)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4.5.2: Time measure for Page-Rank computation between three algorithms
4.5.3 For 4 Million URLs

Listed below is the table that represents the time taken by each algorithm for the Page-Rank computation of 4 Million URLs

<table>
<thead>
<tr>
<th>Step</th>
<th>Cluster Rank (in seconds)</th>
<th>Source Rank (in seconds)</th>
<th>Truncated PageRank (in seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node-In BVGraph</td>
<td>967</td>
<td>967</td>
<td>N/A</td>
</tr>
<tr>
<td>Node-Out BVGraph</td>
<td>909</td>
<td>909</td>
<td>909</td>
</tr>
<tr>
<td>Source-In BVGraph</td>
<td>N/A</td>
<td>14</td>
<td>N/A</td>
</tr>
<tr>
<td>Rank calculation (per iteration)</td>
<td>2520</td>
<td>21</td>
<td>295</td>
</tr>
</tbody>
</table>

Table 4.5.3: Time measure for Page-Rank computation between three algorithms
### 4.5.4 Graph Representation For Time Measure (in Sec)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>URLs: 633061</th>
<th>URLs: 289503</th>
<th>URLs: 4 M</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node InLinks</td>
<td>2905183</td>
<td>21781790</td>
<td>2834647</td>
</tr>
<tr>
<td>Average InLinks per Node</td>
<td>4.6</td>
<td>78.06</td>
<td>5.82</td>
</tr>
<tr>
<td>Clusters:</td>
<td>48271</td>
<td>164136</td>
<td>256919</td>
</tr>
<tr>
<td>Cluster InLinks:</td>
<td>983579</td>
<td>18210270</td>
<td>9120926</td>
</tr>
<tr>
<td>Average InLinks per Cluster</td>
<td>16.35</td>
<td>109.35</td>
<td>32.54</td>
</tr>
<tr>
<td>Sources:</td>
<td>425</td>
<td>14892</td>
<td>482</td>
</tr>
<tr>
<td>Source InLinks:</td>
<td>75217</td>
<td>9988138</td>
<td>509693</td>
</tr>
<tr>
<td>Average InLinks per Source</td>
<td>176.98</td>
<td>670.8</td>
<td>1057.45</td>
</tr>
</tbody>
</table>

| Cluster Rank (Time is directly proportional to Cluster InLinks) | 422 | 6780 | 2520 |
| Source Rank (Time is directly proportional to Source InLinks) | 3 | 660 | 21 |
| Truncated PageRank (Time is directly proportional to number Node InLinks) | 2 | 12 | 17 |

Table 4.5.4: Time measure between algorithms per iteration (in Seconds)
Line Graph (Y-Axis: Time in seconds, X-axis: Node Graph Size)

Bar Graph (X-Axis: Time in seconds, Y-axis: Node Graph Size)
Line Graph (Y-Axis: Time in seconds, X-axis: Cluster InLinks)

Line Graph (Y-Axis: Time in seconds, X-axis: Source InLinks)
Line Graph (Y-Axis: Time in seconds, X-axis: Node InLinks)
4.5.5  Node In-Link Distribution across Nodes

4.5.5.1 Node In-Link Distribution across Nodes for 300K
4.5.5.2 Node In-Link Distribution across Nodes for 600K

![Diagram showing the distribution of nodes and in-links for 600K nodes.](image-url)
4.5.5.3 Node In-Link Distribution across Nodes for 4M

Distribution of Nodes and InLinks for 4M

# of Node

4500000
4000000
3500000
3000000
2500000
2000000
1500000
1000000
500000
0

# of InLinks

0 1 70 139 208 278 349 425 523 642 751 900 1068 1325 1579 1920 2444 3527 5289 6808 8958 12848 15578
4.5.6 Cluster In-Link Distribution across Clusters

4.5.6.1 Cluster In-Link Distribution across Clusters for 300K
4.5.6.2 Cluster In-Link Distribution across Clusters for 600K

![Graph showing the distribution of clusters and inlinks for 600K.](image-url)
4.5.6.3 Cluster In-Link Distribution across Clusters for 4M
4.5.7 Source In-Link Distribution across Sources

4.5.7.1 Source In-Link Distribution across Sources for 300K
4.5.7.2 Source In-Link Distribution across Sources for 600K

Distribution of Sources and InLinks for 600K

# of Sources

# of InLinks
4.5.7.3 Source In-Link Distribution across Sources for 4M
4.5.8 Time Gain Analysis Between Algorithms

Based on the results obtained from the experiments, it is proven that:

- ClusterRank computational time is directly proportional to the Cluster in-links. This is because, in this algorithm the nodes (URLs) are grouped into Clusters based on the ‘virtual directory and dynamically generated links (containing ?, #)’, and the rank of the Cluster is distributed to the nodes contained in the Cluster. The rank of the Cluster is computed using Google’s PageRank algorithm, by treating the Clusters as nodes.

- SourceRank computational time is directly proportional to the Source in-links. This is because, in this algorithm the nodes (URLs) are grouped into Sources based on the ‘domain’, and the rank of the Source is distributed to the nodes contained in the Source. The rank of the Source is computed using Google’s PageRank algorithm, by treating the Sources as nodes.

- Truncated PageRank computation time is directly proportional to the number of URLs. This is because, in this algorithm, the Truncated PageRank is computed using Google’s PageRank algorithm.

4.6 Quality Measure Between Algorithms

A survey is performed among a group of people to measure the quality of the three algorithms. The survey is performed based on the questions listed in appendix A, using 25 different key words. These 25 keywords were identified by using Google’s tool that is available at: https://adwords.google.com/select/KeywordToolExternal. The purpose of this tool is to identify the relevant keywords for websites based on their content. URLs of multiple universities were considered, while using this tool, to identify relevant search keywords for educational domains.

The quality is measured by following the steps:
• Each person participated in the survey perform a keyword search using 25 keywords against three Search-Engine systems. The first Search Engine system uses the data generated by ClusterRank algorithm, the second one uses the data generated by SourceRank algorithm and third one uses the data generated by Truncated PageRank algorithm.

• The average of the quality points is calculated for each question for different keywords.

• The average of the quality points from all the users calculated for each question from step one.

• Finally, the average of the quality points for all the questions from step two, is calculated for each algorithm.

• Based on result from step three, conclude the better algorithm in terms of quality.

4.6.1 Survey Results

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Quality measure based on the scale 1 to 5 (1 being the best) for 25 keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClusterRank</td>
<td>2.06</td>
</tr>
<tr>
<td>SourceRank</td>
<td>1.65</td>
</tr>
<tr>
<td>Truncated PageRank</td>
<td>2.94</td>
</tr>
</tbody>
</table>

4.7 Conclusion of Experimental Results

From the above results we observed that the Page-Rank computation time depends on the URL link structure (Web graph) and also the algorithm used for the computation.
It has been observed that, for algorithms that use only the URL linking structure such as Truncated PageRank, the Page-Rank computation time is directly proportional to the number of URL in-links.

For algorithms such as SourceRank, where the Page-Rank is calculated for the Source and then is propagated to URLs contained within that Source, the Page-Rank computation time is directly proportional to the number of links between the Sources (Source graph size). The same is true for Cluster Rank algorithm.

It is important to note that the number of URLs has no direct relation with the number of Sources or number of Clusters, as it significantly varies depending on the crawling process. For example, if URLs of multiple domains are crawled, the number of Sources generated will be more. The number of Clusters should always be more than the number of Sources for the same set of URLs. This is because the Source is defined based on ‘domain’ and the Cluster is defined based on ‘virtual directory and dynamically generated pages within the URL’.

Based on the results from the manual survey, the quality of the SourceRank is proven as better algorithm than the ClusterRank and Truncated PageRank.

Based on the results from the time-measure experiments, the SourceRank is proven to take less time compared to ClusterRank. This is because, if the URLs of multiple domains are crawled, the number of Sources generated will be more. The number of Clusters should always be more than the number of Sources for the same set of URLs, as the Source is defined based on ‘domain’ and the Cluster is defined based on ‘virtual directory and dynamically generated pages within the URL’.

To take advantage of both Efficiency & Quality, the Source Rank is proven to be the better algorithm out of the three based on the experiments conducted using the available data.
5 References


[8] Gonzalo Navarro. Compressing Web Graphs like Texts, 


Throttling. 21st IEEE International Parallel and Distributed Processing Symposium 
(IPDPS), http://www-static.cc.gatech.edu/~caverlee/pubs/caverlee07ipdps.pdf, 

propagation and probabilistic counting for link-based spam detection. Technical report”, 
2006, 
http://www.dcc.uchile.cl/~ccastill/papers/becchetti_06_automatic_link_spam_detection_r
ank_propagation.pdf

WebBase: A repository of Web page

6 Appendix A – Survey Questions to Evaluate Algorithm Quality

Perform 25 different key-word searches to measure the quality of the algorithm based on the information displayed to the user that is relevant to the key-word.

6.1 Search Keywords

<table>
<thead>
<tr>
<th>pictures</th>
<th>university</th>
<th>Faculty</th>
<th>stadium</th>
<th>undergraduate</th>
</tr>
</thead>
<tbody>
<tr>
<td>map</td>
<td>admissions</td>
<td>Scholarships</td>
<td>loan</td>
<td>mba</td>
</tr>
<tr>
<td>alumni</td>
<td>computer</td>
<td>Graduate</td>
<td>business</td>
<td>research</td>
</tr>
<tr>
<td>students</td>
<td>technology</td>
<td>Accommodation</td>
<td>campus</td>
<td>vacations</td>
</tr>
<tr>
<td>dean</td>
<td>department</td>
<td>Aid</td>
<td>gpa</td>
<td>parking</td>
</tr>
</tbody>
</table>

6.2 Survey Questions

1. First page accuracy (scale 1 to 5, 1 being the best)

2. Second page accuracy (scale 1 to 5, 1 being the best)

3. Result order on the first page (scale 1 to 5, 1 being the best)

4. Result order on the second page (scale 1 to 5, 1 being the best)

5. Overall, are the important pages showing up early? (scale 1 to 5, 1 being the best)

6. Overall, the percentage in result hits are relevant? (Give a percentage)
7 Appendix B – Software and hardware Environment

Software Environment:

- Fedora Core 4, MySQL server 5.0.26, Perl v5.8.6, Apache 2.2.3, Java 1.5

Hardware Environment:

- Pentium 4, 2.0 GHz CPU and 1 GB RAM
8 Appendix C – Database Scripts

8.1 Database Table/View Setup

CREATE TABLE MediaType (  
doc_type_id TINYINT UNSIGNED NOT NULL auto_increment,  
doc_type VARCHAR(20) NOT NULL,  
PRIMARY KEY(doc_type_id));  

INSERT INTO MediaType SET doc_type="text";
INSERT INTO MediaType SET doc_type="image";
INSERT INTO MediaType SET doc_type="OtherBinary";

CREATE TABLE URL (  
url_id INT UNSIGNED NOT NULL auto_increment,  
url VARCHAR(255) NOT NULL,  
doc_type_id TINYINT UNSIGNED NOT NULL,  
container_url INT UNSIGNED,  
title VARCHAR(255),  
cluster_id INT UNSIGNED,  
source_id INT UNSIGNED,  
in_plink_count INT UNSIGNED,  
in_clink_count INT UNSIGNED,  
INDEX (url(255)),  
FOREIGN KEY(doc_type_id) REFERENCES MediaType(doc_type_id),  
FOREIGN KEY(cluster_id) REFERENCES Cluster(cluster_id),  
FOREIGN KEY(source_id) REFERENCES Source(source_id),  
PRIMARY KEY (url_id) );

CREATE TABLE Crawler (  
url_id INT UNSIGNED NOT NULL,  
crawled_date DATE NOT NULL,  
localfullname VARCHAR(255) NOT NULL,  
size INT UNSIGNED NOT NULL,  
FOREIGN KEY(url_id) REFERENCES URL(url_id));

CREATE TABLE ImageProcessor (  
url_id INT UNSIGNED NOT NULL,  
surrounding_words_before VARCHAR(255) NOT NULL,  
surrounding_words_after VARCHAR(255) NOT NULL,  
processed_date DATE NOT NULL,  
FOREIGN KEY(url_id) REFERENCES URL(url_id));

CREATE TABLE URLLinkStructure (  
link_id INT UNSIGNED NOT NULL auto_increment,  
from_url_id INT UNSIGNED NOT NULL,  
to_url_id INT UNSIGNED NOT NULL,  
anchor_text VARCHAR(100),  
update_date DATE NOT NULL,  
FOREIGN KEY(from_url_id) REFERENCES URL(url_id),  
FOREIGN KEY(to_url_id) REFERENCES URL(url_id),  
UNIQUE INDEX (from_url_id, to_url_id),  
PRIMARY KEY (link_id) );
CREATE TABLE PageLocation (  
id SMALLINT UNSIGNED NOT NULL auto_increment,  
description VARCHAR(32) NOT NULL,  
htmltag VARCHAR(32) NOT NULL,  
weight MEDIUMINT UNSIGNED,  
weight_date DATE,  
PRIMARY KEY(id));

CREATE TABLE TextParser (  
url_id INT UNSIGNED NOT NULL,  
processed_date DATE NOT NULL,  
UNIQUE INDEX (url_id),  
FOREIGN KEY(url_id) REFERENCES URL(url_id));

CREATE TABLE KeyWordWork (  
keyword_id INT UNSIGNED NOT NULL,  
url_id INT UNSIGNED NOT NULL,  
location_id SMALLINT UNSIGNED NOT NULL,  
update_date DATE NOT NULL,  
frequency MEDIUMINT UNSIGNED,  
UNIQUE INDEX (keyword_id, url_id, location_id),  
FOREIGN KEY(url_id) REFERENCES URL(url_id),  
FOREIGN KEY(location_id) REFERENCES PageLocation(id),  
FOREIGN KEY(keyword_id) REFERENCES Dictionary(id));

CREATE TABLE KeyWord (  
keyword_id INT UNSIGNED NOT NULL,  
url_id INT UNSIGNED NOT NULL,  
total_weight INT UNSIGNED NOT NULL,  
total_weight_date DATE NOT NULL,  
UNIQUE INDEX (keyword_id, url_id),  
FOREIGN KEY(url_id) REFERENCES URL(url_id),  
FOREIGN KEY(keyword_id) REFERENCES Dictionary(id));

CREATE TABLE Dictionary (  
id INT UNSIGNED NOT NULL auto_increment,  
word VARCHAR(32) NOT NULL,  
UNIQUE INDEX (word(32)),  
PRIMARY KEY (id));

CREATE TABLE Cluster (  
cluster_id INT UNSIGNED NOT NULL,  
base_url VARCHAR(255) NOT NULL,  
cluster_rank FLOAT ZEROFILL,  
cluster_rank_date DATE,  
out_link_count INT UNSIGNED,  
in_link_count INT UNSIGNED,  
cal_last_update DATE,  
cal_reserved_by VARCHAR(255),  
cal_current_iter SMALLINT UNSIGNED,  
old_cr1 FLOAT ZEROFILL,  
old_cr1_date DATE,  
old_cr2 FLOAT ZEROFILL,  
old_cr2_date DATE,  
prop_sec_cluster_id INT UNSIGNED,  
PRIMARY KEY (cluster_id),
FOREIGN KEY(prop_sec_cluster_id) REFERENCES SecondLvlClusterWork(sec_cluster_id)
);

CREATE TABLE SecondLvlClusterWork (sec_cluster_id INT UNSIGNED NOT NULL, sec_base_url VARCHAR(255) NOT NULL, graph_density FLOAT ZEROFILL, PRIMARY KEY (sec_cluster_id));

CREATE TABLE PageRankByCluster (url_id INT UNSIGNED NOT NULL, c_date DATE, c_prc FLOAT ZEROFILL, old_date1 DATE, old_prc1 FLOAT ZEROFILL, old_date2 DATE, old_prc2 FLOAT ZEROFILL, UNIQUE INDEX (url_id), CONSTRAINT FOREIGN KEY(url_id) REFERENCES URL(url_id) ON DELETE CASCADE ON UPDATE CASCADE);

CREATE OR REPLACE VIEW vwURLLinkSource AS SELECT ufrom.source_id fromsource, uto.source_id tosource FROM URLLinkStructure, URL ufrom, URL uto WHERE ufrom.source_id is not NULL AND uto.source_id is not NULL AND URLLinkStructure.from_url_id = ufrom.url_id AND URLLinkStructure.to_url_id = uto.url_id AND ufrom.source_id <> uto.source_id;

CREATE TABLE Source (source_id INT UNSIGNED NOT NULL, base_url VARCHAR(255) NOT NULL, source_rank FLOAT ZEROFILL, source_rank_date DATE, out_link_count INT UNSIGNED, in_link_count INT UNSIGNED, cal_last_update DATE, cal_reserved_by VARCHAR(255), cal_current_iter SMALLINT UNSIGNED, old_cr1 FLOAT ZEROFILL, old_cr1_date DATE, old_cr2 FLOAT ZEROFILL, old_cr2_date DATE, prop_sec_source_id INT UNSIGNED, PRIMARY KEY (source_id), FOREIGN KEY(prop_sec_source_id) REFERENCES SecondLvlSourceWork(sec_source_id));

CREATE TABLE PageRankBySource (url_id INT UNSIGNED NOT NULL, c_date DATE, c_prc FLOAT ZEROFILL,
old_date1 DATE,
old_prc1 FLOAT ZEROFILL,
old_date2 DATE,
old_prc2 FLOAT ZEROFILL,
UNIQUE INDEX (url_id),
CONSTRAINT FOREIGN KEY (url_id) REFERENCES URL (url_id)
ON DELETE CASCADE ON UPDATE CASCADE
);

alter table Cluster ADD INDEX (base_url);
alter table Source ADD INDEX (base_url);
alter table SecondLvlClusterWork ADD INDEX (sec_base_url);
8.2 Database Table/View Cleanup

DROP VIEW vwURLLinkCluster;
DROP VIEW vwURLLinkSource;
DROP TABLE MediaType;
DROP TABLE URL;
DROP TABLE Crawler;
DROP TABLE ImageProcessor;
DROP TABLE URLLinkStructure;
DROP TABLE PageLocation;
DROP TABLE TextParser;
DROP TABLE KeyWord;
DROP TABLE KeyWordWork;
DROP TABLE Dictionary;
DROP TABLE PageRankByCluster;
DROP TABLE SecondLvlClusterWork;
DROP TABLE Cluster;
DROP TABLE PageRankBySource;
DROP TABLE Source;
commit;
Appendix D – Software Setup

WebGraph:

WebGraph is a java package used to represent the graph compactly.

Can be obtained from URL: http://webgraph.dsi.unimi.it/

Java:

Java is used as the programming language to use the WebGraph package.

Can be obtained from URL: http://www.java.com/en/download/linux_manual.jsp

jdbc::mysql:

To update the MySQL database tables with the Page Rank information in the Truncated PageRank module written in Java

Can be obtained from URL: http://dev.mysql.com/downloads/connector/j/5.1.html

Inline-java:

This is a Perl library to access the java modules from Perl.

Can be obtained from URL: http://search.cpan.org/CPAN/authors/id/P/PA/PATL/Inline-Java-0.52.tar.gz

Perl:

Perl (version 5.8.8) is used as the programming language. A fast interpreter, its features to handle and manipulate strings and relatively small memory signatures of its modules make it an ideal language for this project.

Can be obtained from URL: http://www.perl.com/download.csp

MySQL:

The database is designed and implemented using MySQL v3.23.58 and v4.1.1. MySQL is free, scalable and Perl has a rich API for MySQL.
Can be obtained from URL: [http://dev.mysql.com/downloads/](http://dev.mysql.com/downloads/)

Apache:

Apache server v2.0.54 is used for the machines to communicate using the CGI module.

Can be obtained from URL: [http://httpd.apache.org/](http://httpd.apache.org/)
10 Appendix E – Using the Search Engine

All the experiments were performed using machine with ip address 128.198.144.16. The modules are made available at:

- /home/padipudi/webgrap/truncated-pagerank-1.0

The 600K and 4M URL datasets were obtained from sonali_new database on 128.198.144.19.

The textparser.pl was run on 128.198.144.19 and the database tables were copied on to 128.198.144.16 to run the page-rank computation modules.

The Apache instance that serves up all the three Page-Rank systems is available at:

- http://128.198.144.16:1180/cgi-bin/search.pl

The cgi-bin virtual directory that Apache uses to serve up the pages, is available at:

- /home/padipudi/apache/cgi-bin.