An Online Version Of

Hyperlink-Induced Topic Search (HITS) Algorithm

A Writing 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

December 2010

SAN JOSÉ STATE UNIVERSITY

The Undersigned Writing Project Committee Approves the Writing Project Titled

AN ONLINE VERSION OF

HYPERLINK-INDUCED TOPIC SEARCH (HITS) ALGORITHM

by

Amith Kollam Chandranna

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

Dr. Chris Pollett, Department of Computer Science	12/16/2010
Dr. Mark Stamp, Department of Computer Science	12/16/2010
Dr. Tsau Young Lin, Department of Computer Science	12/16/2010

ABSTRACT

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

by Amith Kollam Chandranna

Generally, 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" (Kleinberg, 1996) operates in an offline mode to perform page rank calculation. In this project, we have implemented an online mode of page ranking for this algorithm. This will improve the overall performance of the Search Engine.

This report describes the approach used to implement and test this algorithm. Comparison results performed against other existing search engines are also presented in this report. This is helpful in describing the efficiency of implemented algorithm.

ACKNOWLEDGEMENTS

I would like to thank Dr. Chris Pollett for his excellent guidance throughout this project work and my committee members, Dr. Mark Stamp and Dr. Tsau Young Lin, for their time and effort. Also, a special thanks to my family and friends for their support.

Table of Contents

List of Figures

List of Tables

1. Introduction

Hyperlink-Induced Topic Search (HITS) algorithm (Kleinberg, 1996) 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 have implemented an online ranking algorithm which approximates HITS.

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 the file system. 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 the following: 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 have implemented 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 ranking algorithm. Hence, the existing HITS algorithm has been modified to provide this new functionality.

In Section 2, how the Hyperlink-Induced Topic Search (HITS) algorithm works will be discussed. Section 3 describes an existing search engine, Yioop! (Pollett, 2009). Yioop! is

a GPLv3, open-source, PHP search engine. This project uses Yioop! search engine to implement our version of HITS algorithm. Yioop! is being developed by Dr. Chris Pollett. Section 4 describes the implementation details of modified HITS algorithm inside Yioop! Search engine. In Section 5, a comparison of results for our newly implemented search algorithm is compared against existing search engines. Finally, the conclusion provides the strengths and weaknesses of the implemented search algorithm.

2. PageRank Algorithm

This section describes the working of PageRank Algorithm. PageRank is the link analysis algorithm used by Google search engine. The below steps describe the pseudo code of PageRank algorithm:

Step 1: Download the web (this is the offline part)

Step 2: Assign each downloaded page an initial page rank of 1/(number of pages downloaded).

Step 3: Compute rounds until the sum of the differences squared of ranks between rounds is small. Do:

Step 3a: Set the page rank of a page u for round (i+1) to be

$$PR(u) = \sum_{v \in B_u} \frac{PR(v)}{L(v)}$$

where

L is the number of outlinks from node v, B_u is the set of nodes adjacent to u, and $PR_i(v)$ is the page rank of v in the ith round.

Step 4: Use the PR_i's of the last round calculated as the final page rank.

PageRank algorithm calculates the ranks of the web pages iteratively until desired level of convergence is achieved. The computation method used to calculate ranks is called the “Power method”. PageRank was one of the earliest search engine algorithms introduced on the World Wide Web. Analyzing the working details of this search algorithm is essential in implementing our online HITS algorithm

3. Hyperlink-Induced Topic Search (HITS) algorithm

Hyperlink-Induced Topic Search (HITS) algorithm is one of the ranking algorithms for web page used by search engines. It was developed by Jon Kleinberg, a Computer Science Professor at Cornell University. This algorithm calculates the ranking of web pages in an offline mode. Teoma ("Teoma Web Search", n.d.) is one such search engine that uses HITS algorithm to rank web pages. Teoma is now acquired by Ask.com.

3.1 Implementation details of the existing HITS algorithm

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 are 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 have implemented the online version of this algorithm.

The pseudo code for HITS algorithm ("Google's PageRank and Beyond", 2006) 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,

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

Step 2: $y^{(k)} = L x^{(k)}$

Step3: $k = k + 1$

Step 4: Normalize $x^{(k)}$ and $y^{(k)}$

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

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

$y^{(k)} = L L^T y^{(k-1)}$ → Equation 2

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.

3.2 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.

3.3 Deliverable 1

The first phase of this project involved implementing the existing HITS algorithm in PHP. A static block of 10 x 10 nodes were considered for implementing this algorithm. Even though this implementation was far from representing the actual “world wide web”, it gave a framework for developing the HITS algorithm. The idea was to then extend this to create the final

implementation of HITS algorithm. This final version will be implemented using Yioop! Search Engine (to be discussed in the next section)

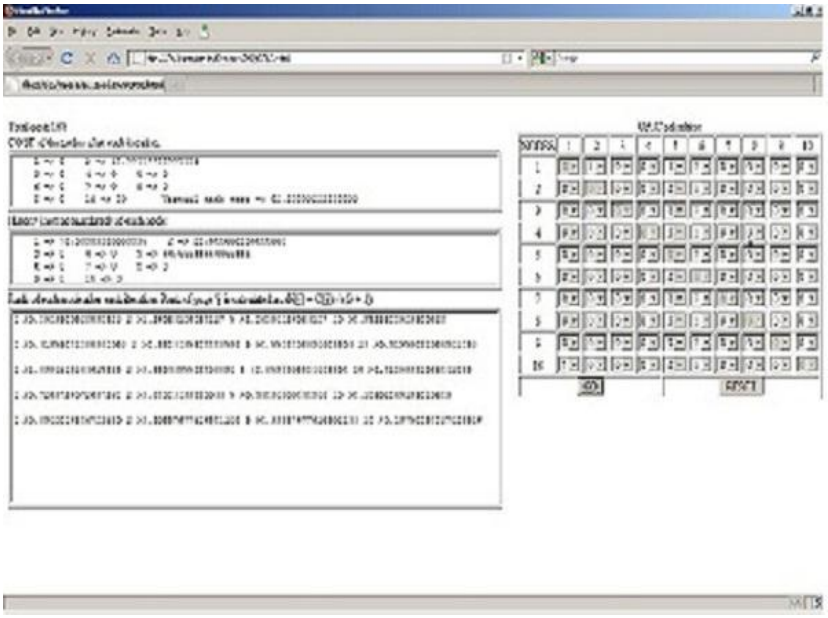

Figure 1: A simulation of original HITS algorithm
Source: HITS (n.d.)
http://localhost/hits.html

4. Yioop! Search Engine

Yioop! search engine (Pollett, 2009) is a GPLv3, open-source, PHP search engine. This is developed by Dr. Chris Pollett. Since this search engine is based on open source licensing it was favorable to implement our version of modified HITS algorithm

4.1 System Architecture

The Yioop! Search Engine is based on the Model-View-Controller (MVC) pattern. It uses a variation of the Online Page Importance Calculation (OPIC) Algorithm ("Adaptive On-Line Page Importance Computation", 2003) to crawl and rank pages. It is written in PHP and the system requirements for its execution are mentioned below:

1. A web server
2. PHP 5.3 or better
3. Curl libraries for downloading web pages.

To be a little more specific Yioop! has been tested with Apache 2.2; however, it should work with other web servers. For PHP, you need a build of PHP that incorporates multi-byte string (mb_ prefixed) functions, Curl, Sqlite, the GD graphics library and the command-line interface. Below is the directory structure of Yioop! Source code:

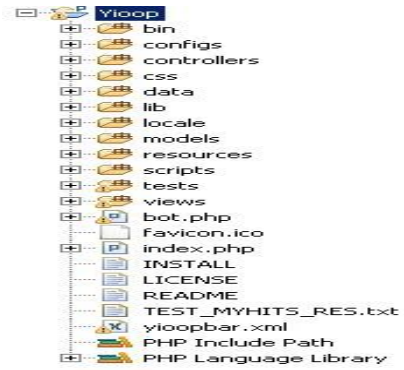

Figure 2: Directory structure of Yioop! Search Engine
Source: "SeekQuarry" (2009)
http://www.seekquarry.com/

The scripts “queue_server.php” and “fetcher.php” (located under “bin” directory) are executed from the command-line. Yioop! offers the flexibility to run multiple fetcher process. They can also be configured to run from different machines. This allows the crawling process to fetch more web pages in a short period of time.

In order to successfully implement the changes to the ranking algorithm in Yioop! v0.42, it is necessary to under its structure and internal workings. The below details describe the directory structure and the source code associated with Yioop! v0.42. The Yioop search engine consists of three main scripts (“Yioop! Documentation”, n.d.):

bin/fetcher.php

Used to download batches of urls provided the queue_server.

bin/queue_server.php

This maintains a queue of urls that are going to be scheduled to be seen. It also keeps track of what has been seen and robots.txt info. Its last responsibility is to create the index_archive that is used by the search front end.

index.php

This web page acts as the search engine web page. It is also used to handle message passing between the fetchers (multiple machines can act as fetchers) and the queue_server.

The file index.php is essentially used when you browse to an installation of a Yioop! website. The description of how to use a Yioop! web site is given in the sections starting from the Yioop! “User Interface section”. The files fetcher.php and queue_server.php are only connected with web “crawling”. If one already has a stored crawl of the web, then you no longer need to run or use these programs. For instance, you might obtain a crawl of the web on your home machine and upload the crawl to your ISP hosting your website with an

instance of Yioop! running on it. This website could serve search results without making use of either fetcher.php or queue_server.php. To perform a web crawl you need to use both of these programs however as well as the Yioop! web site. This is explained in detail in the section Managing Crawls.

The Yioop! folder itself consists of several files and sub-folders. The file index.php as mentioned above is the main entry point into the Yioop! web application. yioopbar.xml is the xml file specifying how to access Yioop as an Open Search Plugin. favicon.ico is used to display the little icon in the url bar of a browser when someone browses to the Yioop! site. A URL to another file bot.php is given by the Yioop! robot as it crawls websites so that website owners can find out information about who is crawling their sites. Here is a rough guide to what the Yioop! folder's sub-folder contain:

bin

This folder is intended to hold command line scripts which are used in conjunction with Yioop!

configs

This folder contains configuration files. You will probably not need to edit any of these files directly as you can set the most common configuration settings from with the admin panel of Yioop! The file config.php controls a number of parameters about how data is stored, how and how often the queue_server and fetchers communicate, and which file types are supported by Yioop! createdb.php can be used to create a bare instance of the Yioop! database with a root admin user having no password. This script is not strictly necessary as the database should be creatable via the admin panel. The file default_crawl.ini is copied to WORK_DIRECTORY after you set this folder in the admin/configure panel. There it is

renamed as crawl.ini and serves as the initial set of sites to crawl until you decide to change these.

controllers

The “controllers” folder contains all the controller classes used by the web component of the Yioop! search engine. Most requests coming into Yioop! go through the top level index.php file. The query string then says who is responsible for handling the request. In this query string there is a part which reads c= ... This says which controller should be used. The controller uses the rest of the query string such as the arg= variable to determine which data must be retrieved from which models, and finally which view with what elements on it should be displayed back to the user.

css

This folder contains the style sheets used to control how web page tags should look for the Yioop! site when rendered in a browser

data

This folder contains a default sqlite database for a new Yioop! installation. Whenever the WORK_DIRECTORY is changed it is this database which is initially copied into the WORK_DIRECTORY to serve as the database of allowed users for the Yioop! system.

lib

This folder is short for library. It contains all the common classes for things like indexing, storing data to files, parsing urls, etc. lib contains two main subfolders: processors and index_bundle_iterators. The processors folder contains processors to extract page summaries for a variety of different mimetypes. The index_bundle_iterator folder contains a variety of

iterators useful for iterating over lists of documents which might be returned during a query to the search engine.

locale

This folder contains the default locale data which comes with the Yioop! system. A locale encapsulates data associated with a language and region. A locale is specified by an IETF language tag. So for instance, within the locale folder there is a folder en-US for the locale consisting of English in the United States. Within a given locale tag folder there is a file configure.ini which contains translations of string ids to string in the language of the locale. This approach is the same idea as used in Gettext .po files. Yioop's approach does not require a compilation step or a restart of the web server for translations to appear. On the other hand, it is slower than the Gettext approach, but this could be easily mitigated using a memory cache such as memcached or apc. Besides the file configure.ini, there is a statistics.txt file which has info about what percentage of the id's have been translated. Finally, although not used in the default Yioop! system. It is possible for a given locale folder to have a sub-folder pages with translation of static pages used by a Yioop! installation.

models

This folder contains the subclasses of Model used by Yioop! Models are used to encapsulate access to secondary storage. i.e.,, accesses to databases or the file system. They are responsible for marshalling/de-marshalling objects that might be stored in more than one table or across several files. The models folder has within it a data sources folder. A data source is an abstraction layer for the particular file system and database system that is being used by a Yioop! installation. At present, data sources have been defined for sqlite, sqlite3, and mysql databases.

resources

This is used to store binary resources such as graphics, video, or audio. For now, just stores the Yioop! logo.

scripts

This folder contains the Javascript files used by Yioop!

tests

This folder contains UnitTest's for various lib components. Yioop! comes with its own minimal UnitTest class which is defined in the lib/unit_test.php.

views

This folder contains View subclasses as well as folders for elements, helpers, and layouts. A View is responsible for taking data given to it by a controller and formatting it in a suitable way. Most Views output a web page; however, some of the views responsible for communication between the fetchers and the queue_server output serialized objects. The elements folder contains Element classes which are typically used to output portions of web pages. For example, the html that allows one to choose an Activity in the Admin portion of the website is rendered by an ActivityElement. The “helpers” folder contains Helper subclasses. A Helper is used to automate the task of outputting certain kinds of web tags. For instance, the OptionsHelper when given an array can be used to output select tags and option tags using data from the array. The layout folder contains Layout subclasses. A Layout encapsulates the header and footer information for the kind of a document a View lives on. For example, web pages on the Yioop! site all use the WebLayout class as their Layout. The WebLayout class has a render method for outputting the doctype, open html tag, head of the

document including links for style sheets, etc. This method then calls the render methods of the current View, and finally outputs scripts and the necessary closing document tags.

In addition, to the Yioop! application folder, Yioop! makes use of a WORK DIRECTORY. The location of this directory is set during the configuration of a Yioop! installation. Yioop! stores crawls, and other data local to a particular Yioop! installation in files and folders in this directory. In the event that you upgrade your Yioop! installation you should only need to replace the Yioop! application folder and in the configuration process of Yioop! tell it where your WORK DIRECTORY is. Of course, it is always recommended to back up one's data before performing an upgrade. Within the WORK DIRECTORY, Yioop! stores three main files: profile.php, crawl.ini, and bot.txt. Here is a rough guide to what the WORK DIRECTORY's sub-folder contain:

cache

The directory is used to store folders of the form ArchiveUNIX_TIMESTAMP, IndexDataUNIX_TIMESTAMP, and QueueBundleUNIX_TIMESTAMP. ArchiveUNIX_TIMESTAMP folders hold complete caches of web pages that have been crawled. These folders will appear on machines which are running fetcher.php. IndexDataUNIX_TIMESTAMP folders hold a word document index as well as summaries of pages crawled. A folder of this type is needed by the web app portion of Yioop! to serve search results. These folders can be moved from machine to whichever machine you want to server results from. QueueBundleUNIX_TIMESTAMP folders are used to maintain the priority queue during the crawling process. The queue_server.php program is responsible for creating both IndexDataUNIX_TIMESTAMP and QueueBundleUNIX_TIMESTAMP folders.

data

If a sqlite or sqlite3 (rather than say MySQL) database is being used then a seek_quarry.db file is stored in the data folder. In Yioop!, the database is used to manage users, roles, locales, and crawls. Data for crawls themselves are NOT stored in the database.

log

When the fetcher and queue_server are run as daemon processes log messages are written to log files in this folder. Log rotation is also done. These log files can be opened in a text editor or console app.

schedules

This folder has three kinds of subfolders: IndexDataUNIX_TIMESTAMP, RobotDataUNIX_TIMESTAMP, and ScheduleDataUNIX_TIMESTAMP. When a fetcher communicates with the web app to say what it has just crawled, the web app writes data into these folders to be processed later by the queue_server. The UNIX_TIMESTAMP is used to keep track of which crawl the data is destined for. IndexData folders contain mini-inverted indexes (word document records) which are to be added to the global inverted index for that crawl. RobotData folders contain information that came from robots.txt files. Finally, ScheduleData folders have data about found urls that could eventually be scheduled to crawl. Within each of these three kinds of folders there are typical many sub-folders, one for each day data arrived, and within these subfolders there are files containing the respective kinds of data.

It is necessary to understand the documentation related to Yioop! 0.42. This is because in order to implement our version of HITS algorithm, the internal workings of Yioop! should be clear. This includes the directory structure, the implementation of the priority queues, the

structure of the inverted index, graphical user interface created by the “views”, the workings of the search interface, etc.

4.2 Deliverable 2

This deliverable involved setting up the environment to execute Yioop! v0.42. The following are the system requirements necessary to setup Yioop:

1. XAMPP (Windows version 1.7.3)
2. Windows 7 (x64)

XAMPP (“apache friends – xampp”, n.d.) includes Apache web server, PHP, Curl modules and other modules that are essential for the execution of Yioop!. XAMPP comes bundled with all these components in one package, so this simplifies the process of installation and configuration of all the components.

The below image shows the search page of the Yioop! search engine:

Figure 3: Yioop! Search Page

Once we login into the Yioop! installation, we can view the admin crawl page. This page will be similar to the one displayed below:

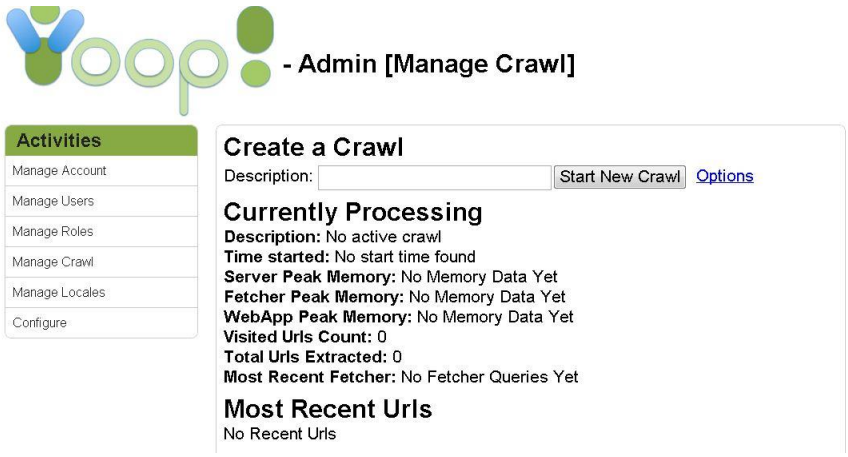

Figure 4: Yioop Admin Crawl Page

The seed sites used to initiate the crawl are configured by clicking on the "Options" (as observed in Figure 4). This web page is as shown below:

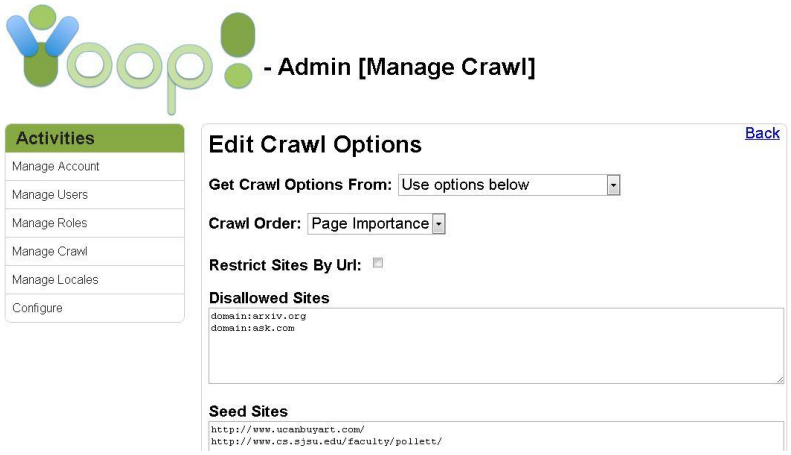

Figure 5: Yioop! Seed Sites

The previous crawls performed in Yioop! are shown in the admin page. They are listed as shown below:

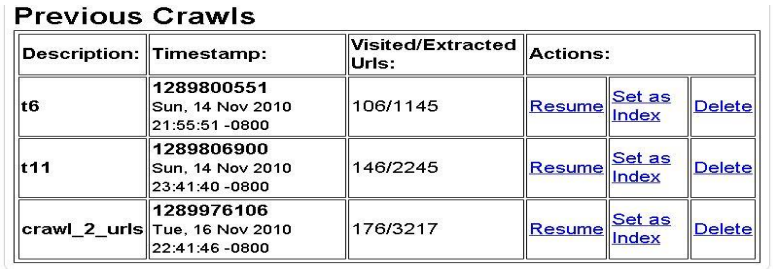

Figure 6: Yioop! Previous Crawls

We can initiate a new crawl by typing the name for the crawl in the admin crawl page, as shown below:

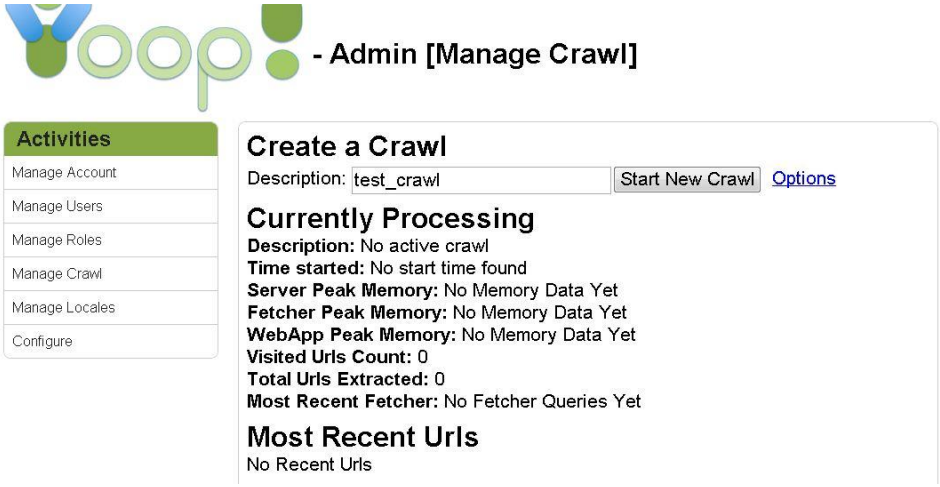

Figure 7: Yioop! Initiate new crawl

Once a crawl has been initiated, we can see the progress of the crawl on the same admin page. This is as shown below:

Figure 8: Yioop! Crawl Status

The admin page allows setting new password for the current account. This page is as shown below:

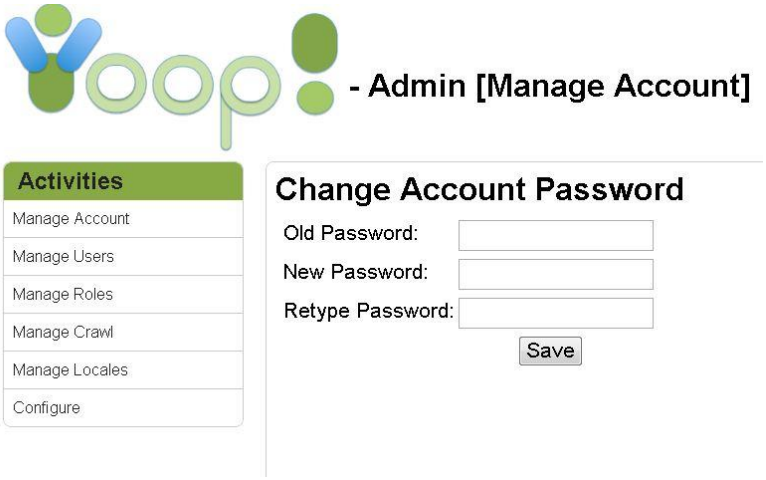

Figure 9: Yioop! Change Password

Also, new users and roles can be added by visiting the pages displayed below:

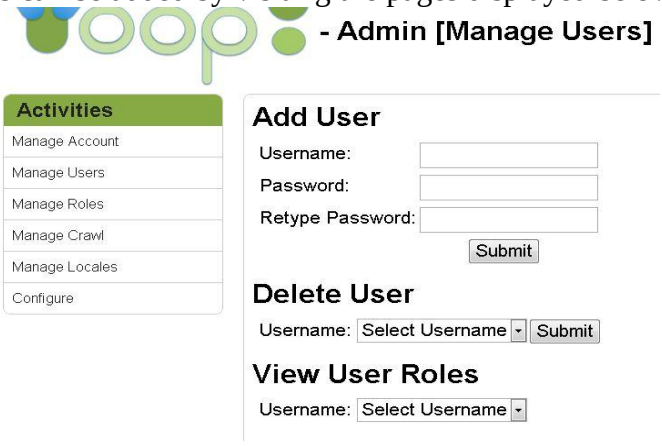

Figure 10: Yioop Manage Users

Figure 11: Yioop! Manage Roles

The “Configure” page describes the main properties of the search engine. The database configurations, queue server settings and the robot details are set on this page. This page is as shown below:

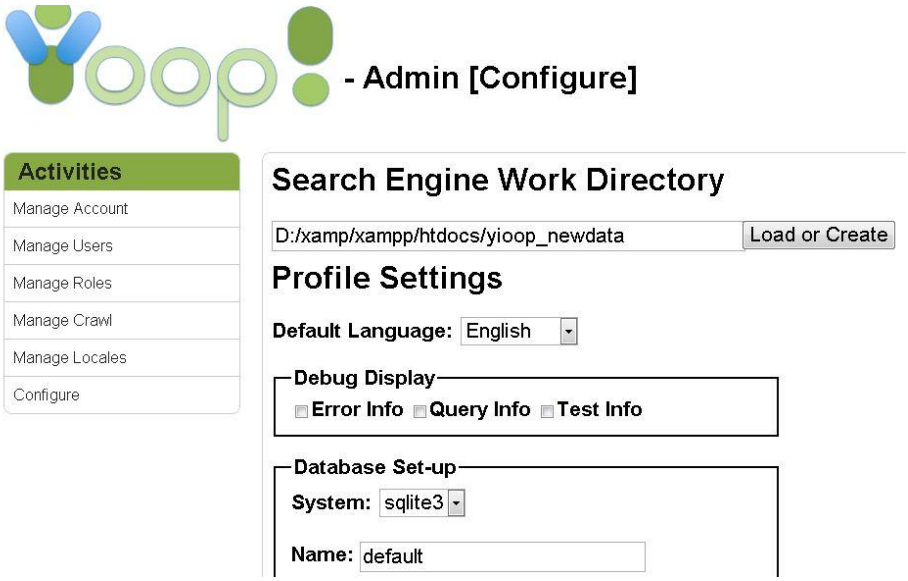

Figure 12: Yioop! Configure page

The locale settings describe the location and language settings. These are essential for the normal functioning of a search engine. This page is displayed as shown below:

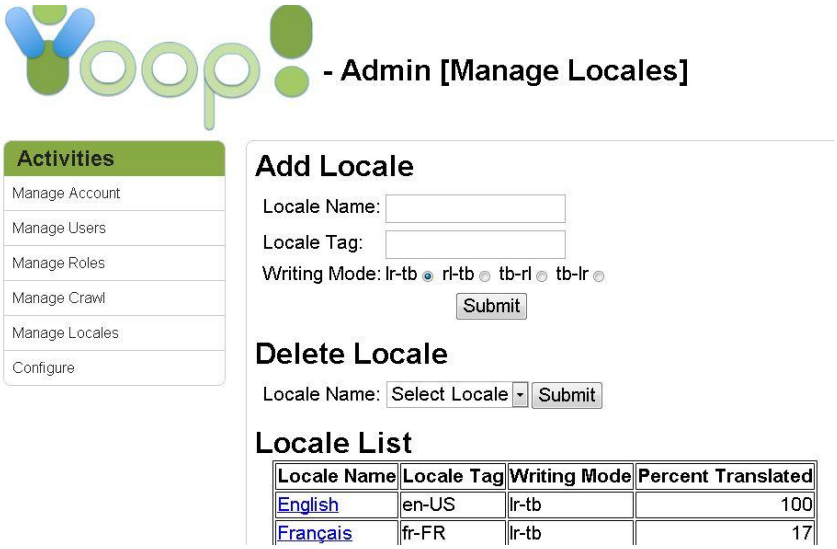

Figure 13: Yioop! Locale page

The Yioop! Settings page describes the configurations to be used during the search process. These include the number of results to be displayed per page, language to be used and the search index to be used. The web page for this displayed as shown below:

Figure 14: Yioop! Settings page

Tests were carried out to make sure that the installation of Yioop! is functioning fine by doing crawls, verifying that the index and queue archive bundles are stored in appropriate directories and finally performing searches for various query terms. All these tests confirmed that the installation of Yioop! v0.42 is working successfully. Now, the next phase was to extend the "Deliverable 1" implementation into the framework of Yioop! mentioned above. This will be discussed in more detail in section 4.

5. Online version of HITS Algorithm

The implementation produced as part of "Deliverable 1" helped in understanding the working of HITS algorithm. This implementation used a graph of 10 x10 nodes to simulate a static network. This helped to create a basic framework that can be used to implement the final modified HITS algorithm.

5.1 Design

The structure of the existing Yioop! v0.42 search engine was thoroughly analyzed. This understanding was crucial for the final implementation of our algorithm. The findings related to these observations are:

The current version of Yioop! uses "Online Page Importance Calculation" (OPIC) algorithm for crawling the pages. It uses a single "priority" queue to maintain the list of urls to be crawled next. Once, a particular page has been crawled, it is marked as "seen" and removed from the queue.

The basic idea in an OPIC algorithm is to start the crawl with initial "cash" distributed to all the seed sites. In each crawl session, the site with the highest cash is crawled. Once a site is crawled, all its cash is equally distributed to its children (i.e., outgoing links). This process continues until the crawling process is terminated. The sites are then sorted in the descending order of the cash accumulated by them. This gives the ranking for each site. The search module then presents the "search results" to the user's queries based on these rankings.

The crawled sites are stored in the "index" bundle on the file system. This is later used to retrieve results during the search process. The "priority" queue is stored in-memory and hence there is no disk access required to gather the site's "cash" during the crawl. The "admin" web page controls the various configurations related to each crawl.

Once a crawled is completed successfully, query terms can be searched on the "search" web page. The search results, together with rankings and other metrics are then displayed to the user. The above listed observations helped in deciding which components/scripts of the Yioop! v0.42 had to be modified in order to implement the online version of HITS algorithm. These observations further helped to create the design specifications and they can be listed as follows:

The Yioop! v0.42 uses OPIC algorithm to crawl web pages. This has to be replaced with an online version of HITS algorithm. The OPIC algorithm in Yioop! uses one "priority" queue to hold the urls that are to be crawled next. But HITS algorithm produces two rankings for each web page i.e., an authority and a hub rank. Hence, we have to implement two queues for gathering both authority and hub ranks for each web page. Even though we have to implement two priority queues, the under-lying functions used to create these queues remain the same. These queues can be conveniently called the "authority" and "hub" queues.

In the initial phase, we store the seed sites into both these queues. Each of them is assigned an initial weight of one. Then, the site (taken from the "hub" queue) with highest cash is crawled. As all the sites have an initial weight of one, the first site in the queue can be considered as the one having the highest cash.

Once a particular site (from "hub" queue) has been crawled, its weight will be divided equally among its outgoing links. This cash distribution process will be handled by the "fetcher.php" script. Now, the weight of the site (whose cash we just distributed to its children), will be reset to zero. Please note, that the cash will be distributed to the outgoing links which reside in the "authority" queue. This process is depicted more clearly in the below diagram:

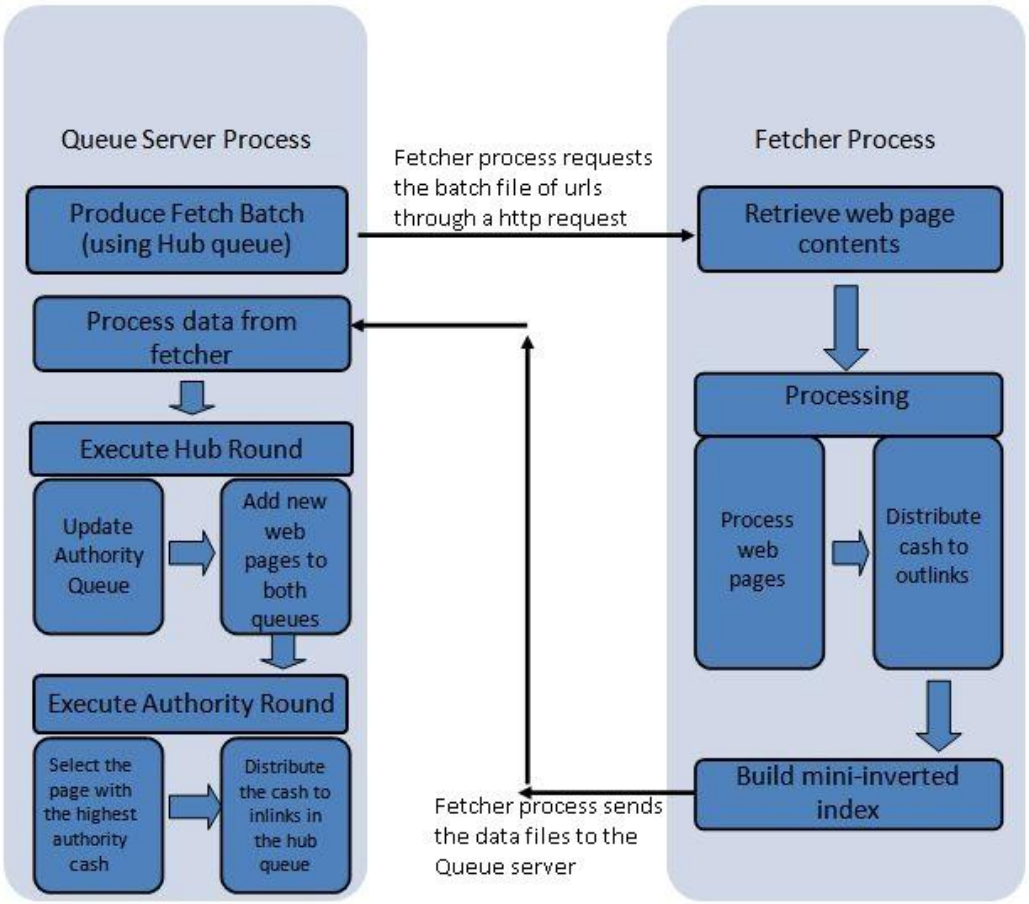

Figure 15: Overview of Queue Server and Fetcher process

The above diagram depicts "one" iteration of the crawling process. This process is repeated continuously until a sufficient number of web pages are crawled or until terminated by the user. This sequence is initiated by the "Produce Fetch Batch" in the Queue server. This "batch" file contains the list of urls to be downloaded by the Fetcher process. These urls are chosen from the "hub" queue.

The priority queues i.e., hub and authority queues, are maintained in the Queue Server process. Once the data sent the Fetcher(s) is received by the Queue Server, it is processed and the main index and the priority queues are updated accordingly. Queue Server and the Fetcher can be configured to run on a single host or can be executed on distributed systems. Also, multiple Fetchers can be configured to run simultaneously. This speeds up the crawling process and

increases the overall efficiency of the system. The advantage of this design is that the script running the Fetcher process needs no modification, as the priority queues are maintained in the Queue Server.

5.2 Priority Queues Normalization

Normalization is a process which scales up or down the weights of the urls in the priority queues. The factor by which this normalization process is applied to each urls is decided based on the overall queue weight. In the current configuration, if the overall queue weight goes below "MIN_QUEUE_WEIGHT" of 1/100000, then the normalization process is applied to that particular queue.

The process of normalization followed in Yioop! can be described as follows:

Normalization process (for a particular queue) is initiated if the overall queue weight goes below "MIN_QUEUE_WEIGHT" of 1/100000.

If the total weight of the queue is greater than zero, the new weight for a given url is calculated as:

Given: current_url_weight, total_weight and new_total

Where new_total = "NUM_URLS_QUEUE_RAM" (This constant is set in the configuration file and describes the maximum number of urls that will be held in ram [as opposed to in files] in the priority queue. Currently, its value is set to "300000")

New weight of the given url = (new_total * current_url_weight) / total_weight

If the total weight of the queue is less than or equal to 0, the new weight for a given url is calculated as:

New weight of the given url = (new_total / count), where "count" indicates the total number of urls existing in the queue.

The normalization process is essential for the priority queues. This is because during the crawl process as new urls are added (and existing weight of the urls are updated) constantly. This in turn causes the total weight of the queue to exceed/decrease with no limit on upper/lower bounds. This has two disadvantages: First, as there is no upper/lower bound on the queue weight, there is a possibility of data overflow. Second, the rate of convergence of the overall algorithm is unpredictable and would take longer to converge. Hence, the normalization process helps to overcome both these issues.

5.3 Convergence

In general, the page crawling strategies can be categorized into following three cases ("Adaptive On-Line Page Importance Computation", 2003):

1. Random: We choose the next page to crawl randomly with equal probability.
2. Greedy: We read the page with the highest cash.
3. Cycle: We choose some fixed order and use it to cycle around the set of pages.

Below diagram shows the rate of convergence for the "crawl strategies" discussed above:

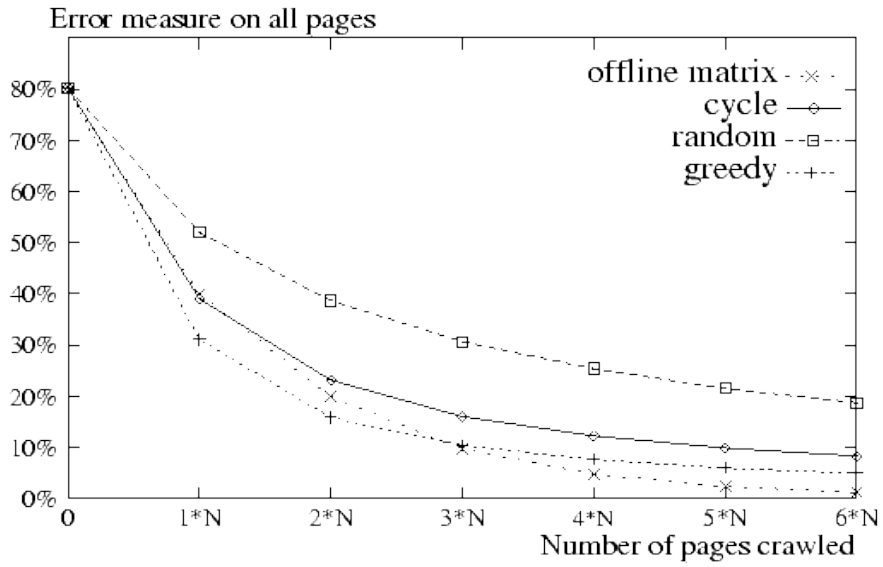

Figure 16: Convergence of OPIC
Source: Adaptive On-Line Page Importance Computation. (2003)
http://www2003.org/cdrom/papers/refereed/p007/p7-abiteboul-img107.gif

The current implementation of Yioop! Search engine (v0.42) uses "greedy" method of selecting the next web pages for crawling (based on cash). Hence, the rate of convergence is fastest (according to the results in Figure 4) compared to other crawling strategies.

The original implementation of HITS algorithm (Kleinberg, 1996) involves two main steps ("Google's PageRank and Beyond", 2006): "First, a neighborhood graph N related to the query terms is built. Second, the authority and hub scores for each page in N are computed. So, essentially the HITS algorithm iteratively calculates the hub and authority scores using the Iterative Power Method. It is to be noted that HITS algorithm with normalization always converges" ("Google's PageRank and Beyond", 2006).

Now, our implementation of online HITS algorithm mimics the behavior of OPIC algorithm in terms of "crawling strategy". This is because we maintain two priority queues in the Queue Server i.e., the hub and authority queues. For crawling purposes, we always select the URL with the highest cash from the hub queue. As this is a "greedy" method of crawling web pages, the rate of convergence is proved to be the fastest (based on the results in figure 4). The cash distribution and normalization processes are applied to both the queues in our implementation. The below points will describe these in more detail (Please refer figure 3):

1. The "Produce Fetch Batch" process selects the urls with the highest cash (from the hub queue) into a "fetch batch" file. This is requested by one of the fetcher processes through a http request. Fetcher(s) retrieve the web page content based on this "fetch batch" file.
2. After this, the fetcher(s) processes/parses the web page contents to create a "mini-inverted" index based on the downloaded web page contents. This is transferred to the Queue Server.
3. The arrival of the "mini-inverted" index (sent by a fetcher) initiates the hub and authority round (in that order) in the Queue Server.

4. In the “Hub” round, the weights of the pages that were provided to the fetcher (as part of the fetch batch file) are distributed equally to their “outlinks” (in the authority queue) and weights of the original pages are reset to 0 (in the hub queue).
5. In the “Authority” round, the weights of the pages (highest weights in the authority queue) is distributed to their “inlinks” in the hub queue.
6. The successful completion of both hub and authority rounds signal the end of one iteration in the crawling process.

Hence, it can be proved that our implementation of HITS algorithm will always converge (i.e., after a fixed number of iterations)

5.4 Pseudo code for implementing “online” HITS algorithm

We can implement the “online” HITS algorithm using an open source engine framework. In our project, we have used Yioop! 0.42 to implement the algorithm. The below steps are mentioned with respect to the server process which will coordinate the crawling activities between the fetchers. In case of Yioop!, the script “queue_server.php” acts as the server process. Also, no modifications are necessary to the existing fetcher script present as part of Yioop! 0.42.

Step 1: Implement two priority queues sorted in decreasing of their weight values.

Step 2: Initialize, the priority queues with the seed sites (with each one of them assigned a weight of 1).

Step 3: Pick the top N web urls from the hub priority queue. Then download these “N” web pages. The same web page might get downloaded more than once (during different iterations). Urls once added to the queues are never removed, but their weights get updated accordingly (as mentioned in Steps 4 and 5).

Step 4: The weights of the downloaded web pages (retrieved from the hub queue) are equally distributed to their outgoing links in the authority queue. The weights of the downloaded web pages are then set to 0 in the hub queue. The urls (present in the set of downloaded pages) which are not present in both the priority queues are marked as new. These new ulss are then added to both the queues. The weights of all these newly added urls are set to 0 in both the priority queues.
Step 5: The urls with maximum weight from the authority queue are chosen. This weight is equally distributed to its incoming links in the hub queue. The weights of all the chosen urls from the authority queue are then set 0.
Step 6: The hub and the authority priority queues are then normalized.
Step 7: Repeat steps 3 to 6 until terminated by the user.
Step 8: The values present in the authority and hub queues after step 7 represent the authority and hub ranks respectively.

5.5 Implementation

For testing purposes, we have crawled approximately around 30000 pages. Below are the screen shots of the test results shown for our implementation of HITS algorithm, Yioop! v0.42 and Nutch v1.0.

We have used the seed sites in all of the cases mentioned below. The below screenshots are just to provide a high-level view of the testing scenarios carried in our project. Later in this report, the results from these will be analyzed in more detail. This comparison will give a good understanding about the efficiency of our current implementation of HITS.

All these search engines were implemented on Windows 7 Operating System. The specific details related to Nutch search engine implementation are provided below (For details related to Yioop! implementation, please refer to Section 3).

The below code snippets describe some of the modifications carried out.

In queue_server.php, we need to create a new priority queue to hold the "Authority" web pages. This is performed by the following piece of code:

```
$this->web_queueA = new WebQueueBundle(
            CRAWL_DIR.'/cache/'.self::queue_base_name.'A'.
            $this->crawl_time, URL_FILTER_SIZE,
            NUM_URLS_QUEUE_RAM, $min_or_max);
```

where "web_queueA" indicates that it is the "Authority" queue. This is implemented in form of a queue bundle, which is the data structure provided by the "WebQueueBundle.php". The other parameters to this function call include the directory location of the web queue bundle, the time of the crawl, the URL filter size, the number of maximum urls supported in the priority queue and the kind of crawling strategy used.

The below code sets appropriate permissions on the queue created above:

```
$this->db->setWorldPermissionsRecursive(
            CRAWL_DIR.'/cache/'.self::queue_base_name.'A'.$this->crawl_time);
```

After the priority queue is created successfully, we have the new "Authority" priority queue ready for use. The operations that can be performed on this priority queue include: adding new page urls, adjusting the weight of the existing urls, deleting the urls, moving the urls up/down in the priority queue (based on their ranks), etc.

The config.php controls the various settings of the search engine. In this configuration file, we can set various parameters like NUM_FETCHERS, NUM_URLS_QUEUE_RAM, NORMALIZE_FREQUENCY, PAGE_TIMEOUT, MAX_LINKS_PER_PAGE,

MAXIMUM_CRAWL_DELAY, etc. The values of these settings control the overall behavior of search engine.

We have modified “search_view.php” to display appropriate “Authority” and “Hub” ranks for the displayed results. Also, the results are sorted based on the average of both these ranks.

System requirements of Nutch v1.0

1. Nutch v1.0 installation package
2. Cygwin
3. Apache Tomcat v6.0

Steps for installing Nutch:

1. Unzip the Nutch package to Cygwin home folder.
2. Check if the installation works fine by typing the below command in the Cygwin command-prompt:

   ./bin/nutch

   Please verify that the below output is displayed.

Figure 17: Displaying the Nutch help file

3. Set the “CLASSPATH” and “JAVA_HOME” variables appropriately to reflect the path to Lucene core and Java JDK installations.

4. Verify that the above system variables are set correctly by typing this command at the Cygwin command-prompt:

   ./bin/nutch crawl

   The below output will be displayed if these variables are set correctly:

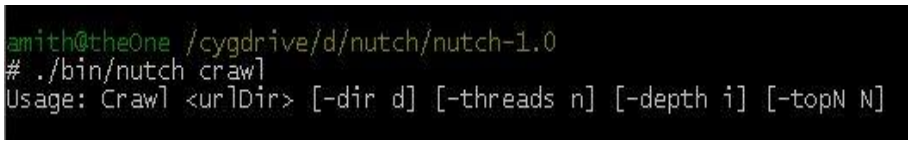

Figure 18: Nutch Crawl Options

5. To configure things for intranet crawling you must, create a directory with a flat file of seed urls. These flat file would contain the seed sites that need to be crawled.
6. Now, you need to edit the conf/crawl-urlfilter.txt. In this file replace "MY.DOMAIN.NAME" with the name of the domain you wish to crawl.
7. Also, we need to edit conf/nutch-site.xml. Here, we need to set the appropriate values for properties like "http.agent.name", "http.agent.description", "http.agents.robots", etc.
8. Once, the configuration files are edited, we can initiate the crawl process. We can start the crawl by typing this command at the Cygwin command-prompt:

   bin/nutch crawl urls -dir crawl -depth 3 -topN 50

   where dir - specifies the download location. The crawled web page contents are stored to this location.

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

   topN - determines the maximum number of pages that will be retrieved at each level up to the depth.
9. After the "crawl" is done, we can perform search by putting the nutch war file into your servlet container. The web application finds its indexes in the "crawl" directory, relative to where you start Tomcat, so use a command like:

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

10. Now, the search page can be reached by typing the below url in a browser:

    http://localhost:8080/

    Below web page should be displayed:

Figure 19: Nutch Home Page

The below screen shots show the search results displayed for a user query "sjsu math". The same query was executed across all three search engines.

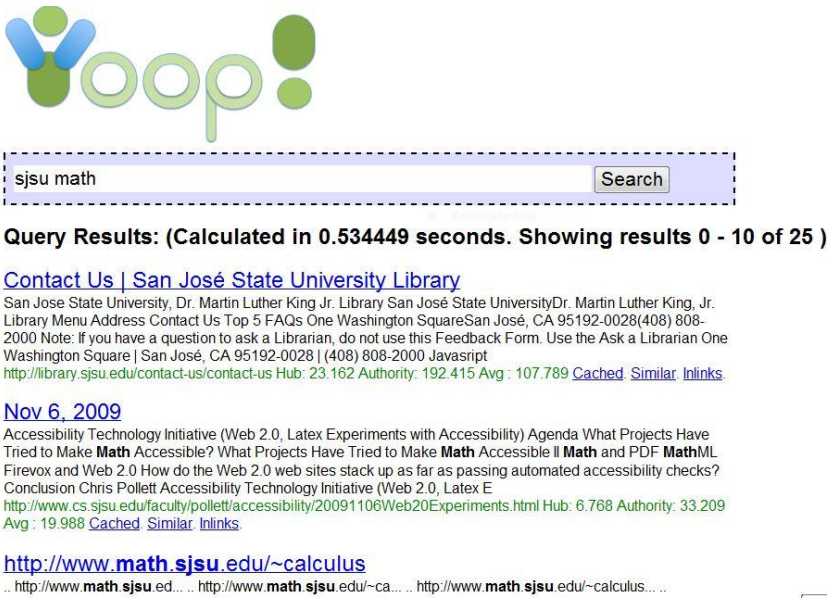

Figure 20: Results from "Online HITS based Search Engine"

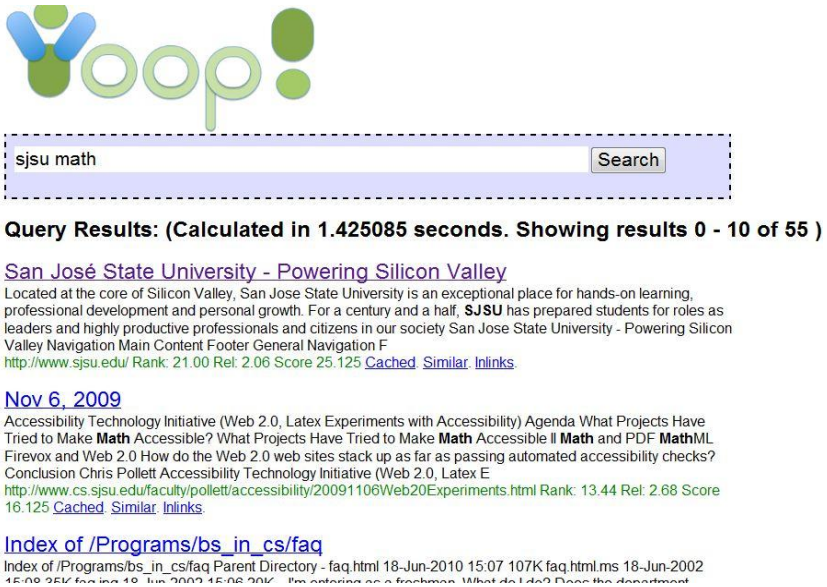

Figure 21: Results from Yioop! v0.42

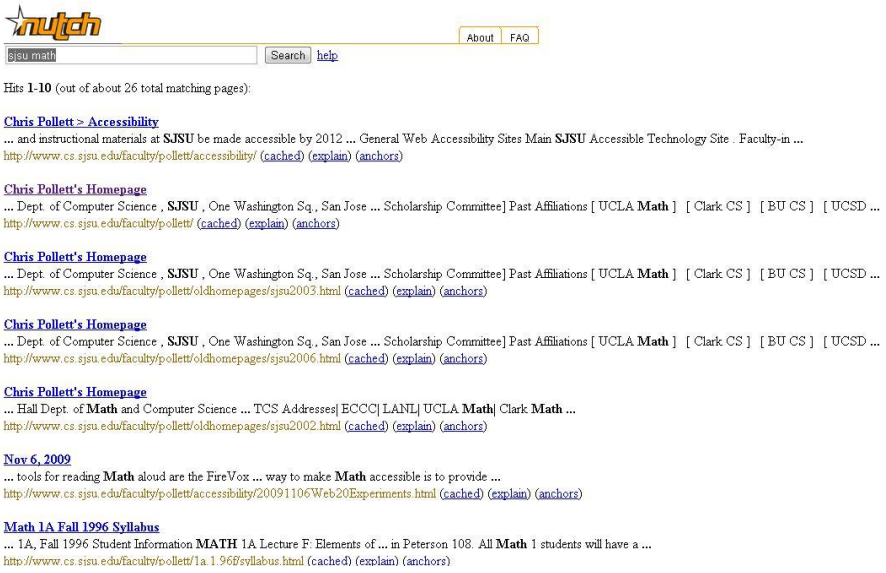

Figure 22: Results from Nutch v1.0

6. Comparison results

The seed sites that were used in the crawl are mentioned below:

1	http://www.sjsu.edu/
2	http://www.physics.sjsu.edu/
3	http://www.cs.sjsu.edu/
4	http://www.sjsu.edu/math/
5	http://www.sjsu.edu/science/
6	http://www.sjsu.edu/academic_programs/
7	http://www.sjsu.edu/colleges_departments/
8	http://www.sjsu.edu/siteindex/
9	http://www.sjsu.edu/resources/links/CampusDirectory/
10	http://www.sjsu.edu/healthscience/
11	http://my.sjsu.edu
12	http://library.sjsu.edu/
13	http://www.cs.sjsu.edu/faculty/pollett/

Table 1: Seed sites

We have used "trec_eval" v8.1 ("Text REtrieval Conference", n.d.) utility to compare the search results obtained from our HITS implementation against Yioop! v0.42 and Nutch.

The installation process for generating this binary file is:

1. Download the trec_eval v8.1 source code package from this url:

   http://trec.nist.gov/trec_eval/index.html
2. Unzip and untar the package. This creates the "trec_eval.8.1" source directory.
3. Open a Cygwin terminal. Make sure the Cygwin installation contains "gcc" and "make" utilities. These are required for compiling the "trec_eval" source code.
4. Change directory (i.e., cd) to the root of this source code.
5. Now type the below command at the command-prompt:

   Make
6. Once, compilation is successful, the "trec_eval.exe" should be created in the current directory.

7. Verify that the compilation is successful by typing the below command at the command-prompt:

   ./trec_eval.exe

   The below output should be displayed:

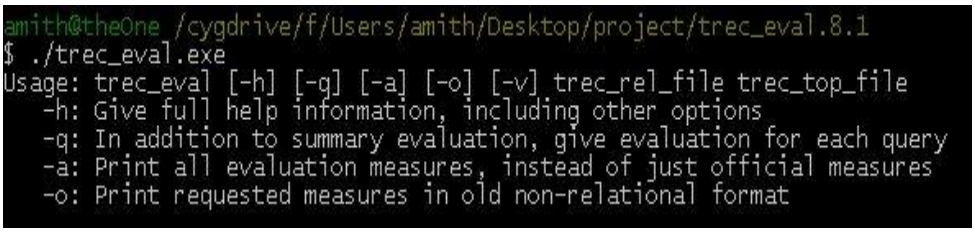

Figure 23: trec_eval Help Output

The format of executing this utility from command-line is:

Usage: trec_eval <trec_rel_file> <trec_top_file>

where "trec_rel_file" contains the relevance judgements, and

"trec_top_file" contains the results that needs to evaluated

So, we have to manually create a list of expected results (for some queries). The results have to be listed in decreasing order of their ranks. This will be the "relevance judgements file". The results returned from each of the search engines (i.e., our implementation of HITS, Yioop v0.42 and Nutch) will be treated as the "trec_top_file". Please refer to the help file for further details about the specific format of these input files.

Below is the list of expected results for each query:

Query	Documents / URLs (in decreasing order)	Rank
sjsu math	http://www.sjsu.edu/math/	1
	http://www.sjsu.edu/math/programs/	2
	http://www.sjsu.edu/math/courses/	3
	http://www.sjsu.edu/math/programs/graduate/	4
	http://www.sjsu.edu/math/programs/undergraduate/	5
	http://sites.google.com/site/mathadvisingsjsu/	6
	http://www.sjsu.edu/math/docs/advisor_list_09-10.pdf	7

	http://www.math.sjsu.edu/~mathclub/	8
	http://www.math.sjsu.edu/camcos/	9
	www.math.sjsu.edu/~calculus/	10
sjsu science	www.science.sjsu.edu/	1
	www.sjsu.edu/healthscience/	2
	www.cs.sjsu.edu/	3
	www.sjsu.edu/polisci/	4
	slisweb.sjsu.edu/	5
	www.science.sjsu.edu/cosac/	6
	www.biology.sjsu.edu/specialprogs/cls/index_cls.aspx	7
	www.sjsu.edu/casa/	8
	www.sjsu.edu/socialsciences/	9
	www.sjsu.edu/depts/socs/	10
pollett	http://www.cs.sjsu.edu/faculty/pollett/	1
	http://www.cs.sjsu.edu/faculty/pollett/grad_co/	2
	http://www.sjsu.edu/people/chris.pollett/	3
	http://www.cs.sjsu.edu/faculty/pollett/masters/	4
	http://www.cs.sjsu.edu/faculty/pollett/174.1.10f/	5
	http://www.cs.sjsu.edu/faculty/pollett/185c.3.10f/	6
	http://www.cs.sjsu.edu/faculty/pollett/oldclasses.shtml	7
	http://www.cs.sjsu.edu/faculty/pollett/masters/?templates.shtml#top	8
	http://www.sjsu.edu/cfd/accessibility/FIR/	9
	http://www.seekquarry.com/	10
sjsu	http://www.sjsu.edu/	1
	http://www.sjsu.edu/academic_programs/	2
	http://www.sjsu.edu/colleges_departments/	3
	http://www.sjsu.edu/alumni_and_community/	4
	http://www.sjsu.edu/students/	5
	http://www.sjsu.edu/resources/links/KingLibrary-AcademicGateway/	6
	http://www.sjsu.edu/resources/links/CampusDirectory/	7
	http://www.sjsu.edu/advising/	8
	http://www.sjsu.edu/schedules/	9
	http://www.sjsu.edu/faso/	10
sjsu computer science	http://www.cs.sjsu.edu/	1

	www.sjsu.edu/advising/links/cs/	2
	http://www.sjsu.edu/mscs/	3
	http://www.sjsu.edu/mscs/program-info/	4
	www.sjsu.edu/ugs/assessment/programs/science/compsci/	5
	www.sjsu.edu/mscs/how-to-apply/	6
	http://cs.sjsu.edu/Programs/minor/minor.html	7
	www.sjsu.edu/mscs/research/projects/	8
	www.sjsu.edu/mscs/research/guidelines/	9
	http://cs.sjsu.edu/jobs.html	10

Table 2: Relevance Judgements (manually ranked results)

We use the "Relevance Judgements" from table 2 to create the "trec_rel_file"

Below is the comparison chart (For the search query "sjsu math"):

Our implementation of HITS			**Yioop! v0.42**		
Measure name	**Query id**	**Value**	**Measure name**	**Query id**	**Value**
num_ret	1	24	num_ret	1	26
num_rel	1	10	num_rel	1	10
num_rel_ret	1	6	num_rel_ret	1	3
map	1	0.1286	map	1	0.0435
R-prec	1	0.2	R-prec	1	0.1
bpref	1	0.6	bpref	1	0.3
recip_rank	1	0.1429	recip_rank	1	0.1667
ircl_prn.0.00	1	0.2727	ircl_prn.0.00	1	0.1667
ircl_prn.0.10	1	0.2727	ircl_prn.0.10	1	0.1667
ircl_prn.0.20	1	0.2727	ircl_prn.0.20	1	0.1429
ircl_prn.0.30	1	0.2727	ircl_prn.0.30	1	0.1429
ircl_prn.0.40	1	0.2727	ircl_prn.0.40	1	0
ircl_prn.0.50	1	0.2727	ircl_prn.0.50	1	0
ircl_prn.0.60	1	0.2727	ircl_prn.0.60	1	0
ircl_prn.0.70	1	0	ircl_prn.0.70	1	0
ircl_prn.0.80	1	0	ircl_prn.0.80	1	0
ircl_prn.0.90	1	0	ircl_prn.0.90	1	0
ircl_prn.1.00	1	0	ircl_prn.1.00	1	0
P5	1	0	P5	1	0
P10	1	0.2	P10	1	0.1
P15	1	0.1333	P15	1	0.0667
P20	1	0.25	P20	1	0.1
P30	1	0.2	P30	1	0.1

P100	1	0.06	P100	1	0.03
P200	1	0.03	P200	1	0.015
P500	1	0.012	P500	1	0.006
P1000	1	0.006	P1000	1	0.003

Table 3: Comparison Chart 1

Observing the details in Table 3, the number of "relevant returned queries" (num_rel_ret) for our implementation of HITS is twice as much as Yioop! v0.42. The search results displayed in our HITS is based on the average of authority and hub ranks for a given page. Based on this observation, it can be inferred the quality of the relevant result in our implementation is better than Yioop! v0.42.

Generally, in an information/text retrieval process, "Recall" and "Precision" are the widely used metrics to evaluate the effectiveness of the results. They can be considered as a way of measuring the accuracy of the displayed results.

"When using precision and recall, the set of possible labels for a given instance is divided into two subsets, one of which is considered relevant for the purposes of the metric. Recall is then computed as the fraction of correct instances among all instances that actually belong to the relevant subset, while precision is the fraction of correct instances among those that the algorithm believes to belong to the relevant subset. Precision can be seen as a measure of exactness or fidelity, whereas recall is a measure of completeness." ("Precision and recall – Wikipedia", n.d.)

So, the comparing the values at various levels of recall and precision will help us to accurately measure the quality of different search results. Recall and Precision can be mathematically stated as follows:

$$\text{Recall} = \frac{|\{\text{set of relevant documents}\} \cap |\{\text{retrieved documents}\}|}{|\{\text{set of relevant documents}\}|}$$

$$\text{Precision} = \frac{|\{\text{set of relevant documents}\} \cap |\{\text{retrieved documents}\}|}{|\{\text{set of retrieved documents}\}|}$$

The interpolated Recall and Precision averages at various levels (0.00, 0.10, 0.20, etc) are better in our implementation of HITS in comparison to Yioop! v0.42. The value of "Mean Average Precision" (map) is also considerably good in our implementation. It is also a similar case with the values displayed for precisions at various values.

Our implementation of HITS			Nutch v1.0		
Measure name	**Query id**	**Value**	**Measure name**	**Query id**	**Value**
num_ret	1	24	num_ret	1	11
num_rel	1	10	num_rel	1	10
num_rel_ret	1	6	num_rel_ret	1	2
map	1	0.1286	map	1	0.04
R-prec	1	0.2	R-prec	1	0.2
bpref	1	0.6	bpref	1	0.2
recip_rank	1	0.1429	recip_rank	1	0.2
ircl_prn.0.00	1	0.2727	ircl_prn.0.00	1	0.2
ircl_prn.0.10	1	0.2727	ircl_prn.0.10	1	0.2
ircl_prn.0.20	1	0.2727	ircl_prn.0.20	1	0.2
ircl_prn.0.30	1	0.2727	ircl_prn.0.30	1	0
ircl_prn.0.40	1	0.2727	ircl_prn.0.40	1	0
ircl_prn.0.50	1	0.2727	ircl_prn.0.50	1	0
ircl_prn.0.60	1	0.2727	ircl_prn.0.60	1	0
ircl_prn.0.70	1	0	ircl_prn.0.70	1	0
ircl_prn.0.80	1	0	ircl_prn.0.80	1	0
ircl_prn.0.90	1	0	ircl_prn.0.90	1	0
ircl_prn.1.00	1	0	ircl_prn.1.00	1	0
P5	1	0	P5	1	0.2
P10	1	0.2	P10	1	0.2
P15	1	0.1333	P15	1	0.1333
P20	1	0.25	P20	1	0.1
P30	1	0.2	P30	1	0.0667
P100	1	0.06	P100	1	0.02
P200	1	0.03	P200	1	0.01
P500	1	0.012	P500	1	0.004
P1000	1	0.006	P1000	1	0.002

Table 4: Comparison Chart 2

The comparison results of Nutch v1.0 and our implementation of HITS are shown in Table 4. These results are similar to the comparison of Yioop! v0.42 and our HITS. The interpolated Recall and Precision averages at various levels (0.00, 0.10, 0.20, etc) are better in our implementation of HITS in comparison to Yioop! v0.42. The value of "Mean Average Precision" (map) is also considerably good in our implementation. It is also a similar case with the values displayed for precisions at various values. Comparison was also conducted for various query terms. The results of those were similar to the observations noted above.

Graphical Comparison of Results

Graphical results are easier to understand than tabulated results. So, we also the present the comparisons graphically if the form of x-y plotted charts. Also, the metrics compared (displayed on x-axis) are: MAP (Mean Average Precision), IRP_0.10(Interpolated Recall and Precision Average at 0.10) and P5(Precision after 5 documents are retrieved).

From the observations in the chart it is clear that our implementation of online HITS has fared better than Yioop! 0.42 and Nutch 1.0. The only exception is that for the query term "sjsu computer science" and "sjsu". So overall it can be concluded that our online HITS algorithm has performed better than the other two search engines.

The query terms used in the below comparison are:

Query 1	"sjsu math"
Query 2	"sjsu science"
Query 3	"pollett"
Query 4	"sjsu"
Query 5	"sjsu computer science"

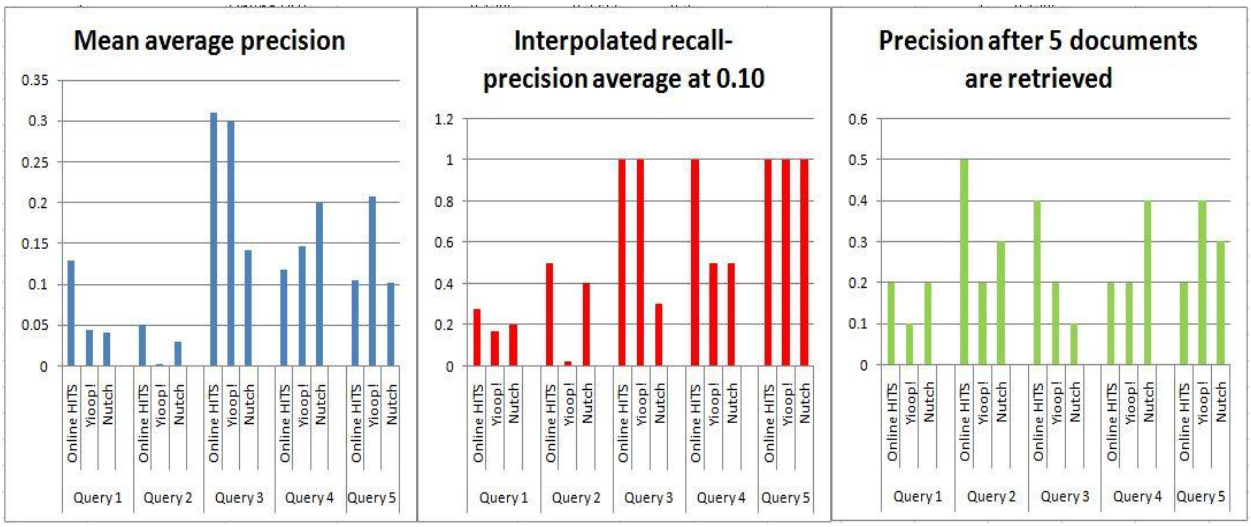

Figure 24: Graphical Comparison Chart

7. Conclusion

In this project, we have implemented 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.

This implementation provides the user with two rankings for each page i.e., hub and authority ranks. This will be helpful to decide the "authorities" or "hubs" for a given topic more accurately.

Most search engines today just sort the results presented to the user based on a single ranking. It will be convenient if they can extend this functionality to sort the results based on hub and authority ranks as well.

Comparisons were carried out to evaluate the performance of our implementation of HITS against Yioop! v0.42 and Nutch v1.0. The tests were carried out for various query terms. From these results, it was observed that our implementation of HITS performed fairly better in terms of delivering relevant results. The "trec_eval" utility published by NIST department was used to evaluate the results set.

Yioop! v0.42 provided a convenient framework to implement these modifications to HITS algorithm. The creators of Yioop! intend to release more memory efficient versions of Yioop! search engine in the future. We plan to port this implementation to future versions of Yioop! as well. This will help to make the implementation of HITS to be more memory efficient.

References

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

Abiteboul, Serge., Preda, Mihai., & Cobena, Grégory. (2003). *Adaptive On-Line Page Importance Computation.* Retrieved Nov 29, 2010 from 2003.org web site: http://www2003.org/cdrom/papers/refereed/p007/p7-abiteboul.html

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

Text REtrieval Conference. (n.d). Retrieved Nov 30, 2010 from TREC home page: http://trec.nist.gov/

Precision and recall – Wikipedia. (n.d). Retrieved Dec 1, 2010 from Wikipedia web page: http://en.wikipedia.org/wiki/Precision_and_recall

Yioop! Documentation. (n.d.) Retrieved Dec 2, 2010 from SeekQuarry web page: http://seekquarry.com/?c=main&p=documentation