DESIGN AND IMPLEMENTATION OF A SEARCH ENGINE WITH THE CLUSTER RANK ALGORITHM

by

YI ZHANG

M.S., Colorado Technical University, 2000

A Thesis Submitted to the Graduate Faculty of the

University of Colorado at Colorado Springs

in Partial Fulfillment of the

requirements for the degree of

Master of Science

Department of Computer Science

2006

This thesis for the Master of Science degree by

Yi Zhang

has been approved for the

Department of Computer Science

by

_____

Dr. Terry Boult

_____

Dr. C. Edward Chow

_____

Advisor: Dr. Jugal K. Kalita

Date _____

ACKNOWLEDGEMENTS

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

I wish to pay special tributes to the fellow engineers, Sonali Patankar and Sunil Bhave, who provided a large set of sample data and make many constructive suggestions.

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

DESIGN AND IMPLEMENTATION OF A SEARCH ENGINE WITH THE CLUSTER RANK ALGORITHM

by

Yi Zhang

(Master of Science, Computer Science)

Thesis directed by Associate Professor Dr. Jugal K. Kalita

Department of Computer Science

Abstract

In this report, I present the design and implementation of a search engine, named Needle. The search engine is designed to handle both text and image search, with a flexible architecture that will facilitate scaling up. Each module is independent of each other and therefore swappable to improve its own functionality without affecting the whole system. More importantly, a new algorithm, Cluster Rank, has been designed and fully implemented to achieve similar goal as that of the existing Page Rank [Brin, 1998] algorithm while providing similar performance and an additional feature for managing similar pages in search result.

TABLE OF CONTENTS

FIGURES

TABLES

CHAPTER 1

INTRODUCTION

Search Engine technology was born almost at the same time as the World Wide Web [Wall 2005a], and has certainly improved dramatically over the past decade and become an integral part of everybody's Web browsing experience, especially after the phenomenal success of Google[1].

At the first glance, it appears that Search Engines have been studied very well, and many articles and theories including the paper by the founders of Google [Brin 1998] have been published to describe and analyze their internal mechanisms. However, in this report the author will demonstrate that there are many unsolved problems in Search Engines as well as unrevealed implementation details, either due to the immaturity of this technology or simply the business nature of their owners.

1.1 The Basic Components of a Search Engine

All search engines includes:

1. A Web crawler.
2. A parser.
3. A ranking system.
4. A repository system.

---

[1] http://www.google.com

5. A front-end interface.

These components are discussed individually below.

The starting point is a Web Crawler (or spider) to retrieve all Web pages: it simply traverses the entire Web or a certain subset of it, to download the pages or files it encounters and save for other components to use. The actual traversal algorithm varies depends on the implementation; depth first, breadth first, or random traversal are all being used to meet different design goals.

The parser takes all downloaded raw results, analyze and eventually try to make sense out of them. In the case of a text search engine, this is done by extracting keywords and checking the locations and/or frequencies of them. Hidden HTML tags, such as KEYWORDS and DESCRIPTION, are also considered. Usually a scoring system is involved to give a final point for each keyword on each page.

Simple or complicated, a search engine must have a way to determine which pages are more important than the others, and present them to users in a particular order. This is called the Ranking System. The most famous one is the Page Rank Algorithm published by Google founders [Brin 1998].

A reliable repository system is definitely critical for any application. Search engine also requires everything to be stored in the most efficient way to ensure maximum performance. The choice of database vendor and the schema design can make big difference on performance for metadata such as URL description, crawling date, keywords, etc. More challenging part is the huge volume of downloaded files to be saved before they are picked up by other modules.

Finally, a front-end interface for users: This is the face and presentation of the search engine. When a user submits a query, usually in the form of a list of textual terms, an internal scoring function is applied to each Web page in the repository [Pandey 2005], and the list of result is presented, usually in the order or relevance and importance. Google has been known for its simple and straight forward interface, while some most recent competitors, such as Ask.com[1], provide much richer user experience by adding features like preview or hierarchy displaying.

1.2 Search Engines Available Today

Other than well-known commercial products, such as Google[2], Yahoo[3] and MSN[4], there are many open source Search Engines, for example, ASPSeek[5], BBDBot[6], Datapark Search[7], and ht://Dig[8]. Evaluating their advantages and disadvantages is not the purpose of this thesis, but based on reviews and feedbacks from other people [Morgan 2004], they are either specialized only in a particular area, or not adopting good ranking algorithms, or have not been maintained for quite a while.

Another important fact is that while most current search engines are focused on text, there is an inevitable trend that they are being extended to the multi-media arena, including dynamic contents, images, sounds and others [Wall 2005b]. None of the open

---

[1] http://www.ask.com
2 http://www.google.com
3 http://www.yahoo.com
4 http://www.msn.com
5 http://www.aspseek.org/
6 http://www.searchtools.com/tools/bbdbot.html
7 http://www.dataparksearch.org/
[8] http://www.htdig.org/

source engines listed above has multimedia searching modules, and none of them is flexible enough to add new ones without significant effort.

1.3 Issues in Search Engine Research

Design of Web crawlers: Web crawler, also known as robot, spider, worm, and wanderer, is no doubt the first part of any search engine and designing a web crawler is a complex endeavor. Due to the competitive nature of the search engine business, there are very few papers in the literature describing the challenges and tradeoffs inherent in web crawler design [Heydon 1999].

Page ranking system: Page Rank [Brin, 1998] is a system of scoring nodes in a directed graph based on the stationary distribution of a random walk on the directed graph. Conceptually, the score of a node corresponds to the frequency with which the node is visited as an individual strolls randomly through the graph. Motivated largely by the success and scale of Google's Page Rank ranking function, much research has emerged on efficiently computing the stationary distributions of Web-scale Markov chain, the mathematical mechanism underlying Page Rank. The main challenge is that the Web graph is so large that its edges typically only exist in external memory and an explicit representation of its stationary distribution just barely fits in to main memory [McSherry 2005].

Repository freshness: A search engine uses its local repository to assign scores to the Web pages in response to a query, with the implicit assumption that the repository closely mirrors the current Web [Pandey 2005]. However, it is infeasible to maintain an

exact mirror of a large portion of the Web due to its considerable aggregate size and dynamic nature, combined with the autonomous nature of Web servers. If the repository is not closely synchronized with the Web, the search engine may not include the most useful pages, for a query at the top of the result list. The repository has to be updated so as to maximize the overall quality of the user experience.

Evaluating the feedback from users: Two mechanisms have been commonly used to accomplish this purpose: Click Popularity and Stickiness [Nowack 2005]. Click Popularity calculates how often a record in the returned list is actually clicked by the user, and promote/demote its rank accordingly. Stickiness assumes the longer an end user stays on a particular page, the more important it must be. While being straightforward, the implementation of these two algorithms can be quite error prone. The data collecting the most difficult part, as the server has to uniquely identify each user. This has been further complicated by the fact that many people want to manually or programmatically promote their own Web sites by exploiting the weaknesses of certain implementations [Harpf 2005].

Two graduate students at UCCS [Jacobs 2005][Kodavanti 2005] have been working on an Image search engine and a text search engine, respectively. Part of their work is to adopt the published Page Rank algorithm [Brin 1998], and the results are quite promising. However, giving the experimental nature of these two projects, they are not suitable for scaling up and not mature enough to serve as a stable platform for future research. A complete redesign and overhaul is needed.

CHAPTER 2

PAGE RANK RESEARCH

2.1 The Original Page Rank algorithm

Google is known for its famous Page Rank algorithm, a way to measure the importance of a Web page by counting how many other pages link to it, as well as how important those page themselves are.

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

PR(A) = (1-d) + d (PR(T1)/C(T1) + ... + PR(Tn)/C(Tn))

In the equation above:

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.

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.

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).

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)`: The (1 – d) 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).

At the first glance, there is a paradox. In order to calculate the PR of page A, one must first have the PR of all other pages, whose Page Rank is calculated in the same way. The algorithm solves it by first assuming all pages to have the same PR of 1, and at each iteration PR is propagated to other pages until all PR stabilize to within some threshold.

Because the large dataset PR algorithm deals with, measuring the stabilization of the PRs can be a difficult job itself. Research indicates that in some cases PR can be calculated in as few as 10 iterations [Haveliwala 1999], or it may take more than 100 iterations [Kamvar 2003].

Another important fact is that when a page does not have outgoing links, the `C(Tn)`, this page becomes a dangling URL, and must be removed from the whole picture. If such “pruning” was not done, the dangling may have critical implications in terms of computation. First, Page Rank values are likely to be smaller than they should be, and might become all zero in the worst case. Second, the iteration process might not converge to a fixed point [Kim 2002].

2.2 Page Rank Calculation Pseudo Code and Complexity

Based on the discussion earlier, we can easily write the pseudo code to calculate the Page Rank. Assume it converges after 40 iterations.

```
Initialize: Set all page to have initial page rank of 1
Loop 40 iterations
   Loop on all pages (M)
      Current page is A
      Get all pages that link to A (N back links)
      Loop on N
         PR(A) = PR(A) + d * PR(Tn) / C(Tn)
      End of loop on back links
      Save PR(A) to database
   End of loop on all pages (M)
End of loop 40 iterations
```

Observing the pseudo code above, there are two nested loops: the outer loop processes all pages (M), and the inner one loops over the number of links (votes) to each page. Therefore the overall complexity can be written as *O(MN)*, where M is the total number of pages and N is the average in-degree (incoming links) on a page.

The process can also be expressed as the following eigenvector calculation, providing useful insight into the concept of Page Rank. Let *M* be the square, stochastic matrix corresponding to the directed graph G of the Web, assuming all nodes in G have at least one outgoing edge. If there is a link from page *j* to page *i*, let the matrix entry m_{ij} have the value $1/N_j$. Let all other entries have the value 0. Also let *Rank(p)* vector represents the importance (i.e., PageRank) of page p. Every iteration corresponds to the matrix-vector multiplication *M times Rank(p).* Repeatedly multiplying *Rank(p) by M* yields the dominant eigenvector *Rank** of the matrix M. Because M corresponds to the stochastic transition matrix over the graph G, Page Rank can be viewed as the stationary

probability distribution over pages induced by a random walk on the Web [Haveliwala 1999].

2.3 Improvements Proposed to Page Rank

Ever since the original Page Rank algorithm was published [Brin 1998], computer scientists have made numerous attempts to improve its performance [Haveliwala 1999] [Arasu 2001] [Chen 2002] [Kim 2002] [Lee 2003] [Kamvar 2003].

The research can be roughly divided into three categories:

- Calculate the dominant eigenvector directly and try to swap certain part of the matrix between main memory and storage media [Haveliwala 1999] [Arasu 2001] [Chen 2002].
- Propose slight modification or improvements over the original algorithm [Kim 2002] [Lee 2003] [Kamvar 2003] [Gupta 2003] to achieve same or similar results in a more efficient way.
- At each step, rather than communicate their current probability values, nodes only the changes in probability value, resulting in much better performance especially during updates [McSherry 2005].
- Parallel calculation in a distributed system [Wang 2004].

These researchers have achieved very impressive results. Kamvar [Kamvar 2003], for example, was able to finish the calculation of 280,000 nodes with 3 million links in about 300 minutes on a single machine without any parallel processing.

The research done by Gupta and Acharyya [Gupta 2003] is very similar to the approach presented in this report and therefore is worth special attention. They modified the graph-independent part of the Page Rank surfer's behavior so that it does not visit pages arbitrarily, instead the surfer is likely to select pages in the same cluster and exhaust them before moving to a different cluster depending on the current page. The rank of a page now depends not only on its in-links but also by the block rank of the cluster to which it belongs; this allows fresh pages to be highly ranked provided they are members of "good clusters".

Their algorithm is summarized in these steps:

- Cluster the graph based on some clustering algorithm. The report did not give any specifics here, but they did mention it is link based, not content based.
- Compute the local Page Rank for each cluster.
- Compute the Block Transition Matrices and the Block Rank Vector
- Merge the result from steps 2 and 3 above to get the final global Page Rank.

They performed experiments on two sets of URLs. One the set is about 30,000 pages in *utexas.edu* domain, and the other is a subset of CORA dataset consisting of the citation graph of 11754 research papers in the "Artificial Intelligence" category [McCallum 2000]. On first set of data, the results produced by the modified algorithm and the original Page Rank differ only by 5%, while on the second set they are

completely different in that the top pages are dominated by the pages with high block rank in the modified algorithm. The performance (wall clock time) results were not given in either of the experiments.

CHAPTER 3

DESIGN AND ANALYSIS OF THE CLUSTER RANK ALGORITHM

3.1 Motivation

After the very first round of design and implementation of the Needle search engine on the UCCS domain, I immediately noticed that two pieces that need improvements. First, the Page Rank calculation always took very long time (roughly 4 hours to process 100,000 pages), and second, the search results were not very pretty from end user perspective in that many consecutive result pages were occupied by similar URLs. For example, in the UCCS domain there are many calendar links[1,2], which present several years of events on a calendar. Naturally all these pages are dynamically generated and they all have almost identical Page Rank. When users search the word "event" or "calendar", they are presented two or three pages of almost identical URLs and other meaningful hits are very difficult to discover.

Famous search engines solve the second problem by grouping them together, and provide a separate link such as "Similar pages" on Google and "More from this site" by Yahoo. However the implementation details are not published due to their business nature.

---

[1] http://tle2.uccs.edu/webcal/month.php
[2] http://easweb.uccs.edu/CS/calendar.php

3.2 Design Goals

Attempting to solve these two problems and inspired by previous research, I designed the Cluster Rank algorithm to speed up the Page Rank calculation while providing the extra feature of grouping similar pages together.

It is not my goal to produce an identical mathematical match to the original Page Rank algorithm, which will be used as comparison baseline nevertheless, but to provide reasonable importance measurements of the URLs as well as a grouping feature that the end user can enjoy.

Keeping the goals in mind, I have taken various intuitive shortcuts and heuristic approaches without mathematical proofs.

3.3 Cluster Rank algorithm

The Cluster Rank algorithm, which is similar to the research done by Gupta and Acharyya [Gupta 2003], has the following steps:

1. Group all pages into clusters.
    a. Perform first level clustering for dynamically generated pages
    b. Perform second level clustering on virtual directory and graph density
2. Calculate the rank for each cluster with the original Page Rank algorithm.
3. Distribute the rank number to its members by weighted average.

Each step is discussed in detail in the following sections.

3.3.1 Clustering

The first choice is to find a good clustering algorithm and it has to be link-based, as parsing each crawled page and finding the similarities among all of them is almost computationally impossible.

3.3.1.1 Traditional clustering algorithms

Several traditionally well-defined clustering algorithms were considered and even briefly experimented, including hierarchical clustering, K-means clustering[Hartigan 1979] and MCL (Markov Cluster Algorithm) [Dongen. 2000]. The most important reason that they were not selected, other than potential performance and complexity reasons, is they are too generic in this giving context. These algorithms cluster generic graph whose nodes and links do not have any inherited meaning. In our problem space, however, the links and the URLs themselves have a lot to tell us. For example, when we see a group of URLs like these:

A. http://www.uccs.edu/~csgi/index.shtml
B. http://office.microsoft.com/en-us/default.aspx
C. http://office.microsoft.com/en-us/assistance/default.aspx
D. http://office.microsoft.com/en-us/assistance/CH790018071033.aspx

We can immediately tell that URL A belongs to the UCCS domain and it has nothing to do with the other three. Without even looking at the links among them, we can make an educated guess that URL A should not be in the same cluster as the other three. In fact they should never belong to the same cluster, no matter how many links are there between them. This is a critical point as our clustering algorithm must not be twisted too easily by a malicious attempt where someone makes numerous links to other important pages from a new page just created.

Similarly, examine the following group:

A. http://tle2.uccs.edu/webcal/month.php?&year=2006&month=10
B. http://tle2.uccs.edu/webcal/month.php?&year=2006&month=09
C. http://tle2.uccs.edu/webcal/month.php?&year=2006&month=08

We can make a reasonable assumption that they are quite similar, again without looking at the links.

The decision of not using generic clustering algorithms was made because they may need a lot of calculation to reveal an already obvious fact, and they have the potential to be misled by excessive links.

3.3.1.2 Non-Link Based Clustering

Previous research [Zamir, 1997] has also explored the possibility of other document content based clustering mechanism. The Word-Intersection Clustering (Word-IC) presented in the research had $O(n^2)$ complexity and it is too slow for users for any Web with substantial size.

3.3.1.3 URL Based Clustering for the Web

Considering the success of the Page Rank system, one definitive factor is that it captured the thinking of a Web page creator --- people make links to other pages only when they think those pages are important. After congregating the thoughts of millions of minds together, the importance of a page is ultimately presented in Page Rank number.

While clustering Web URLs together, we can take advantage of a similar observation by attempting to capture the mind of Webmasters, because more than likely people put similar pages into same virtual directory, or generate them dynamically from same base by given different parameters. This fact is quite simple and other people have

been using it to manage personal navigation space [Takano 2000], or define logical domains [Li 2000a], however it has been ignored by researchers who only look at the Web graph from a pure mathematical perspective.

The intuitive URL based clustering algorithm in this report has two stages:

1) First level clustering: simply cluster all dynamically generated pages together without any calculation;
2) Second level clustering has three steps:
    a. Calculate a set of proposed clusters by grouping URLs in the same virtual directory.
    b. Calculate the graph density of each proposed cluster.
    c. If the density passes a preset threshold *t*, approve the cluster, otherwise, reject it.

3.3.1.4 Clustering threshold

In this particular context, the threshold is concluded from the experimental result: All details and reasoning are discussed later in this report, section 5.5 Choice of Density Threshold.

3.3.2 Calculating the Cluster Rank

In this step, the original Page Rank algorithm is executed at cluster level, as if each cluster is an indivisible unit.

Before applying the published algorithm, preparation is needed --- the link information between individual URLs must be congregated to cluster level. Other than this programming effort, there is nothing theoretical new or challenging.

3.3.3 Distributing Cluster Rank to its members

While other researchers [Gupta 2003] choose to calculate Local Page Rank within the cluster, the distribution algorithm here is again a heuristic one, taking performance as a determinant factor.

Let's review the very spirit of the Page Rank system --- the importance of a page is determined by how many links it receives, and how important the voting pages are. At this stage, the Cluster Rank (importance) is already calculated, and what is required is to distribute them to the member pages based on the number of links. In another straightforward term, the question becomes --- if a cluster has rank of X because it has total of Y incoming links as a group, what is the individual contribution of each member page?

Naturally, weighted average is chosen to answer this question: $P_R = C_R * P_i/C_i$. The notations here are:

P_R : The rank of a member page
C_R : The cluster rank from previous stage
P_i : The incoming links of this page
C_i : Total incoming links of this cluster.

3.3.4 The Impact of Adding New Pages

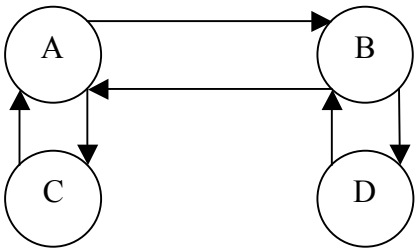
While the Cluster Rank algorithm presented above is quite simple and straightforward, it can prevent a malicious way of promoting Web sites. Website promotion is not the focus of this report, but we can see that simply adding a new page and re-arranging internal structure will not increase the total rank of a Website. Instead this attempt might slightly decrease the rank of each page, as the Cluster rank is already determined at a higher level, and the more pages share it, the smaller piece each page gets.

To further understand the impact of adding new pages, let's see the following example – assume there are two Websites, each of which has two pages (a title page and a content page) linking to each other, and the two title pages also link to each other.

The Page Rank numbers are calculated by WebWorkShop[1], one of many PR calculators available to public.

This extremely simple Web described above and the Page Rank of each URL is illustrated in Figure 1:

[1] http://www.webworkshop.net/Page Rank_calculator.php



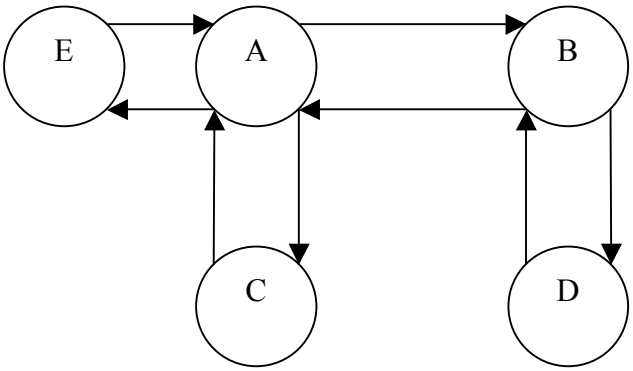
PR of each URL:

A: 1.3
B: 1.3
C: 0.7
D: 0.7

Figure 1. An extremely simple Web and the Page Rank of each URL

These numbers are expected as the title pages (A and B) receive more links than the content page. Because the two websites have exactly the same structure, all numbers are symmetric as well.

When the Webmaster of A and C adds another page E, and makes link to the existing title page A, the first Website grows bigger. The graph and Page Ranks become:



PR of each URL:

A: 1.79
B: 1.23
C: 0.66
D: 0.67
E: 0.66

Figure 2. A new page is added to one Website, but not the other

Now, the title page A of the first Website has higher rank than its counterpart. This is also "as designed" in Page Rank system because:

1) Each page has a minimal Page Rank, including the page just added (E).

2) The more incoming links a page receives, the higher its rank is.

In the example above, page A naturally becomes the most important page of all and notice page B is "downgraded" a little from 1.3 to 1.23.

This behavior is even more obvious when we add yet another new page to the first Website:

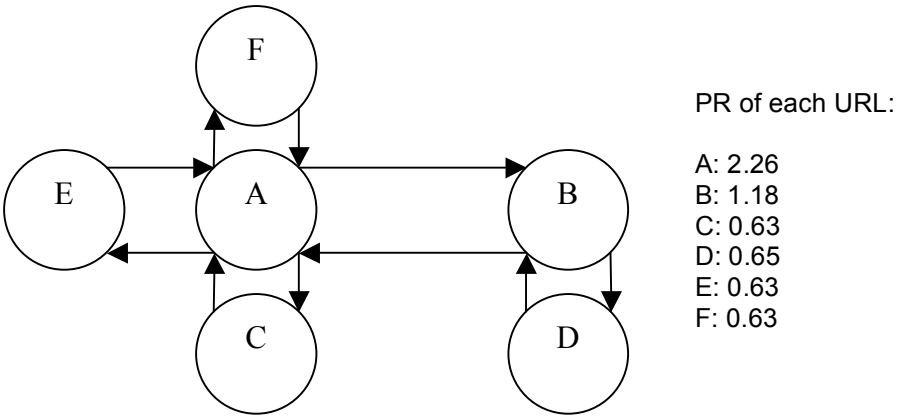


Figure 3. Two new pages are added to one Website, but not the other

Notice the title page B is downgraded even further and it only has about half of the rank of A. The Webmaster of the first site has successfully promoted his site without touching anything outside.

From the perspective of the Page Rank algorithm, everything is quite reasonable as the first Website is much bigger, and it very likely has a lot more to offer. On the real Internet, this way of promoting may not be as effective as in our extremely simplified example because of the sheer volume of existing links and Websites and the way they

intertwine together, however it is still a potential weakness that could be exploited in many ways.

In the Cluster Rank system presented in this report, the starting point is different, but not too far off proportionally:

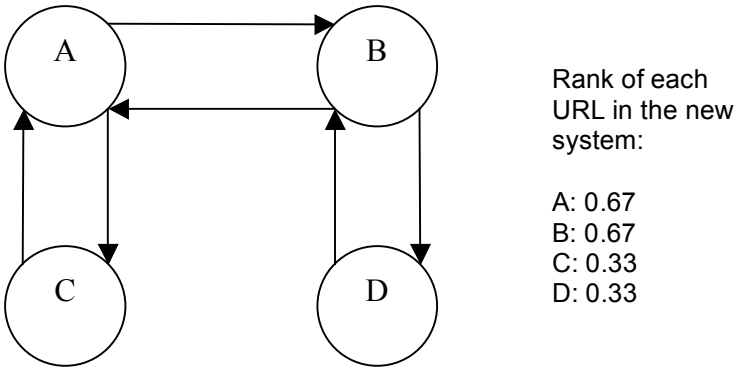


Figure 4. An extremely simple Web and the new ranks

The new ranks are calculated by following the new algorithm presented earlier:

1) Clustering: the two websites both have density of 1, so they are treated as two clusters.

2) Cluster Rank: there are only two clusters linking to each other, therefore each has Cluster Rank of 1.

3) Distribution: Page A receives 2 incoming links, and page C has 1. Therefore the rank of A is 2 * (1/3) = 0.66, and the rank of C is 0.33 and same logic applies to B and D.

An important fact is that although all pages have much smaller number than in the original Page Rank scheme, they still maintain a roughly equal proportional relation at

2:1. The numbers are smaller because there are less “indivisible units”, but if we multiply all ranks by 2 (the average number of pages within a cluster), the end result is rather close to the original algorithm.

After performing the same operation of adding a new page to the first website, the numbers become:

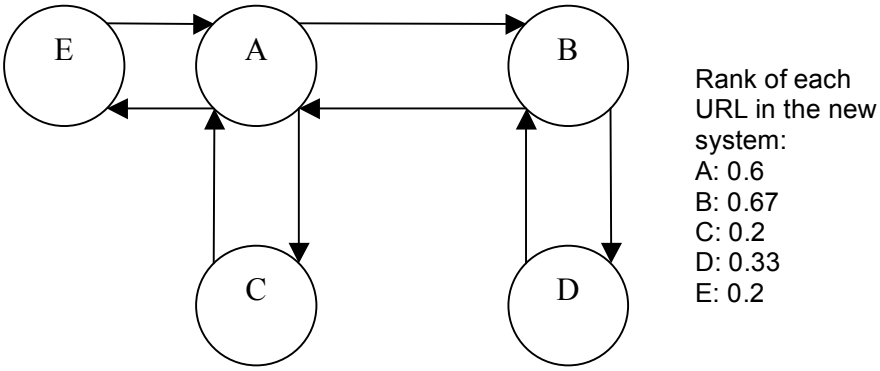


Figure 5. A new page is added to one Website, and the new ranks

Let’s walk through the algorithm again to verify the numbers:

1) Clustering: The first site has density of 0.67 and the second one has 1. Two clusters are generated.

2) Cluster Rank: The same as before. Each cluster has rank of 1.

3) Distribution: Page A receives 3 incoming links, and each of page C and E only has 1. Therefore the rank of A is 3 * (1/5) = 0.6, while C and E each ends up with 0.2. Notice B and D has identical rank as before as nothing changes for the second Website.

Comparing the numbers from Figure 2, we can see the following facts:

- The proportional relation remains the same --- roughly 3:1 between A and its children and 2:1 between B and D.
- B and D are not affected because of this operation.
- All pages from the first website are slightly downgraded because the new page has not drawn any new attention from outside, and it must share the limited resource, the rank of the whole cluster!

One may argue whether the new scheme is more "fair" than the original; however it is quite obvious that if meaningless new pages are added, it does automatically help the whole Website. Instead, a slight penalty is imposed if the new page is not important enough for other Webmasters to add new links to the group.

Here is the result of adding two pages. The same conclusion can be made without going through the routine calculations, shown in Figure 6. Two new pages are added to one Website, and the new ranks.

The analysis above illustrates the benefit of calculating ranks at a higher level and minimizing the impact of individual pages. Again, we must admit that the real Web is much more complicated and adding several pages or even several hundreds of pages may not make any noticeable ripples. A thorough evaluation of the differences of these two algorithms, and how do they perform in various scenarios is far beyond the grasp of casual observations and the scope of this report.

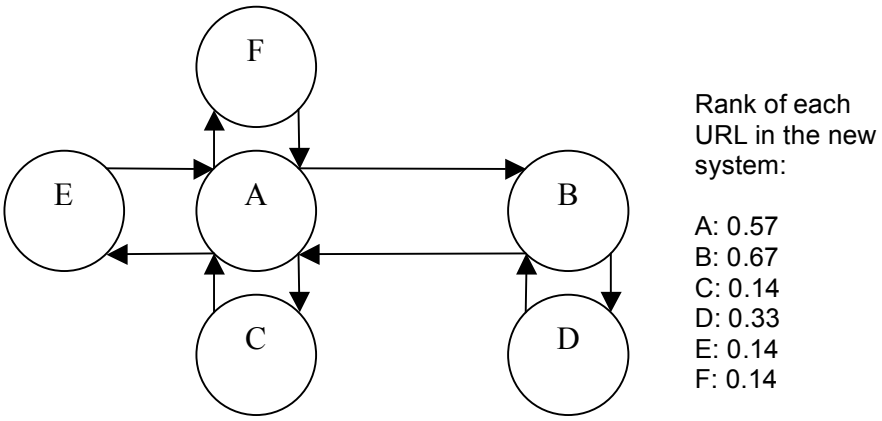


Figure 6. Two new pages are added to one Website, and the new ranks.

3.3.5 The Complexity Analysis

3.3.5.1 First Level Clustering

The purpose of first level clustering is to group all dynamically generated pages and bookmarks together. Its pseudo code looks like:

```
Loop on all pages (M)
   Strip the URL (http://website/main.php?id=123) to get a bare URL
   http://website/main.php
   *Check if the bare URL is in the cluster table
   Yes – Mark the URL to have an existing cluster ID
   No - Insert the bare URL to cluster table to create a cluster and
   mark the URL to have the new cluster ID
End loop
```

The theoretical Algorithm Complexity is $O(M^2)$ because the step of checking the bare URL, marked by (*), could be a linear function on M. However, in practice this step is implemented with a simple SQL SELECT statement, and the search time does not increase in a linear fashion because of the optimization of modern database engines and the pre-built indexes [Gilfillan 2001].

Therefore the complexity is $O(M)$ based on implementation observation.

3.3.5.2 Second Level Clustering

In second level clustering, URLs in the same first level virtual directory are examined to see if they belong to the same cluster. For example, assume the algorithm works on a group of URLs below:

http://business.uccs.edu/careers/careers.html
http://business.uccs.edu/careers/employers.html
http://business.uccs.edu/careers/employmentsites.html
http://business.uccs.edu/careers/events.html

The process is further broken down to three steps.

Step 1

Use the same algorithm as first level clustering, to generate a series of proposed clusters (total of C). The pseudo code is the same, as well as the complexity *O(C)*

Steps 2 and 3

For a directed graph, its density is defined as:

$D = E / (V*(V-1))$)

In the equation above, D is Density; E is the number of edges; V is the number of vertices. If the graph density is higher than a preset threshold, approve the proposed cluster and update all pages with the new cluster id.

Pseudo code:

```
E = 0
Loop on all proposed clusters (C)
   Loop on all pages in the proposed cluster (K)
      Current page is A
      Get all pages that link to A (N back links)
      Count all links from pages inside of the cluster (E_A)
      E = E + E_A
   End of the loop
```

```
   D = E / V^2
   If D > configured threshold
      Approve the cluster and update all pages in it
End of loop on all proposed clusters
```

The theoretical Algorithm Complexity is *O(CKN)*, where C is number of all proposed clusters; K is the average number of pages in a cluster; N is average number of links to a page.

3.3.5.3 Cluster Rank Calculation

The step is nothing more than the original Page Rank algorithm applied at the Cluster level. Therefore the complexity is also *O(CKN)*. See detail analysis earlier in this report, section: 2.2 Page rank calculation Pseudo code and Complexity.

3.3.5.4 Rank Distribution

At this step, weighted average of the Cluster Rank is distributed to all member URLs as their own rank. The pseudo code is as following:

```
Loop on all clusters (C)
   Total incoming links for a cluster L_i = 0
   Loop on all pages in the cluster (K)
      Current page is A, and has (Ai) incoming links
      L_i = L_i + A_i
   End of the loop
   Loop on all pages in the cluster (K)
      Current page is A, and has (Ai) incoming links
      PR'(A) = (Cluster Rank) * Ai / L_i
   End of the loop
End of loop on all clusters
```

The complexity is *O(CK)*, where C is number of all proposed clusters; K is the average number of pages in a cluster.

3.3.5.5 Total Complexity

Adding the time taken by all the substeps together, the total complexity of the new Cluster Rank algorithm is:

$O(M+C+CKN+CKN + CK)$

Obviously, M (total number of pages) equals to C times K, so the complexity becomes $O(M+C+2MN+MN+M)$, where C is total number of clusters, and can be represented as a linear function of M (total number of pages). Therefore the final complexity settles at $O(MN)$.

Casual observation may lead to a conclusion that the new algorithm is the same or slower than the original. However the experiment data indicated that on average there are 2 to 3 pages per cluster and the new algorithm is working on much less number of "units" in the most time consuming step, rank calculation. Fortunately, the time saved here not only covers all other preparation and clustering steps, but also results in a 10% to 20% improvement overall. All details are discussed later in this report, Chapter 5 Test Results and Performance Analysis.

CHAPTER 4

DESIGN OF THE NEEDLE SEARCH ENGINE

4.1 Database Schema

The Needle database consists of 15 tables and two views. The brief definitions and purposes are listed below and grouped by different modules. For details of each tables and view, please see Appendix B, Database Scripts.

- Global tables for all modules:
    - MediaType: To define media types for all modules, such as text, image or other binary files. This table is pre-populated and remains untouched after the database creation.
    - URL: The central reference place for the URLs stored in the Needle search engine. Important fields are:
        - url_id: The unique index of URLs throughout the entire system.
        - url: The complete URL text.
        - in_plink_count: The in-degrees (number of incoming links on a page). The number is stored here to improve performance rather than having to get the count from other tables.
- Crawler tables:

- Crawler: The control table for the crawler. Important fields are:
  - url_id: a foreign key referencing the url_id field in URL table.
  - localfullname: The location of the downloaded page. It will be used by the text parser and the image processor to read the downloaded documents.
  - crawled_date: Keeps track of the exact time this URL is crawled for advanced features like repository refreshing.
- URLLinkStructure: The complete link structure for the Web graph being processed. This table is populated by the Crawler and used extensively by the Ranking system. Important fields are:
  - from_url_id and to_url_id: These are the head and the tail of a link. Two foreign keys referencing the url_id field in URL table.
  - anchor_text: The anchor text is defined as the visible text in a hyperlink[1]. This field is a by-product of crawling and will be used by text parser.

- Ranking system:
  - Page Rank: The only table to store the Page Rank result. Important fields are:

- url_id: a foreign key referencing the url_id field in URL table.
- out_link_count: The out-degree (number of links going out) of a page. This field is populated by crawler as its natural byproduct.
- c_date: current date.
- c_pr: current Page Rank.
- old_date1, old_pr1, old_date2 and old_pr2: two sets of back up of previously calculated Page Rank, for comparison and research purpose.

o Cluster: The master table for Cluster Rank algorithm presented in this report. Important fields are:

- cluster_id: The unique index of a cluster.
- base_url: The bare URL of a cluster.
- cluser_rank: The current Cluster Rank.
- cluster_rank_date: The time when Cluster Rank is calculated.
- out_link_count and in_link_count: The in-degrees and out-degrees of a cluster.

---

[1] http://en.wikipedia.org/wiki/Anchor_text

- old_cr1, old_cr1_date, old_cr2 and old_cr2_date: two sets of back up of previous calculated Cluster Rank, for research purpose.
- prop_sec_cluster_id: a temporary location to store the proposed cluster ID during second level clustering.

- Page RankByCluster: The final calculated rank for each page in Cluster Rank algorithm. Important fields are:
  - url_id: A foreign key referencing the url_id field in URL table.
  - c_prc: The calculated rank of a page in Cluster Rank system.
  - old_date1, old_prc1, old_date2 and old_prc2: two sets of back up of previous data, for research purpose.
- SecondLvlClusterWork: The work table for second level clustering, and saves graph density for each proposed cluster.
- vwURLLinkCluster: A temporary table to congregate links among individual URLs to the cluster level. This is table is created at runtime.

- Text Parser:
    - Dictionary: The dictionary for all words extracted from text documents. The rest parts of the text parser will only reference the keyword IDs to optimize the database performance.
    - PageLocation: The definition table for keyword locations, and their weights.
    - TextParser: The processing record of text parser, saving which URL is processed at which date.
    - KeyWordWork: The work table for text parser to store temporary data during calculation.
    - KeyWord: The final calculation result from text parser. Important fields are:
        - keyword_id: A foreign key referencing the keyword_id field in Dictionary table.
        - url_id: A foreign key referencing the url_id field in URL table.
        - total_weight: the weight of the keyword on the page. This field will be used by front end to retrieve pages.
        - UNIQUE INDEX ((keyword_id, url_id): unique index is enforce to make sure the combination of keyword_id and url_id is unique.

- Image Processor:
    - ImageProcessor: This module is not currently implemented.
- Front end:
    - SearchHistory: To store the search history for statistic purpose. Important fields are:
        - ip_address: The client's IP address.
        - host_name: The client host name, if it can be resolved.
        - request_time: The exact time the search request is received.
        - search_words: The exact words the user is searching.
- Others:
    - vwPRCRComparison: This is a view to compare the Page Rank and Cluster Rank of each page for statistic purpose. This is not a physical table.

4.2 Components

Presented earlier in this report (1.1 The Basic Components of a Search Engine), each search engine must have five basic components: crawler, parser, ranking system, repository system and front-end. The Needle project is no exception.

Flexibility and scalability are the two most important design goals for Needle as it must serve as a common platform for current and future research. Keeping these in mind, all modules are self contained and operate individually. The only dependency is successor modules, such as text parser and ranking system, are expecting the database to be populated in certain way they can process, much like programming interfaces among different functions. Therefore the system architecture is very straight forward, and there is no intertwining of any kind:

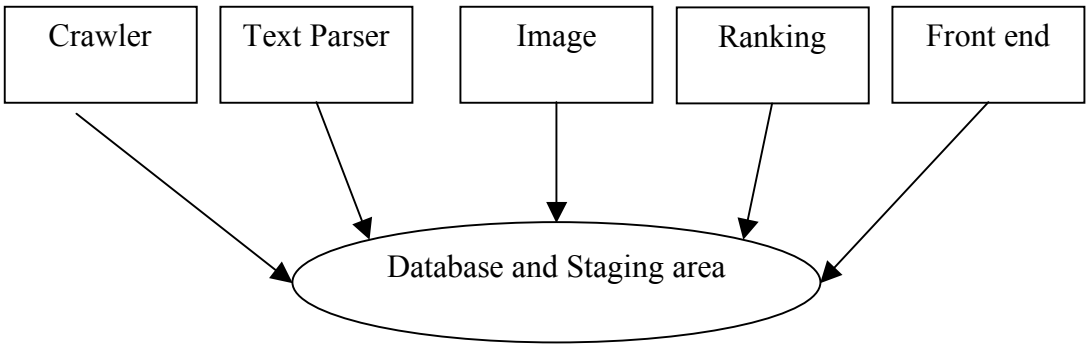


Figure 7. Basic Architecture of the Needle search engine

The flexibility has allowed the author to perform two sets of ranking calculation (the original Page Rank and the new Cluster Rank) in the same system without any change to other modules. And at same time, about five other graduate students in UCCS are also doing their research projects on this architecture without affecting each other.

CHAPTER 5

TEST RESULTS AND PERFORMANCE ANALYSIS

5.1 Sample Data

Calculations are conducted on seven sets of URLs from www.uccs.edu domain, ranging from total of 9,325 URL to 105,674 URLs, and one set of 290,561 URLs from three Colorado universities. These data are all retrieved by the Needle engine crawler in a breadth first fashion.

Data Set ID	Total URLs	Average Links among URLs	Total Clusters	Average Links among Clusters	Average URLs per Cluster
1	9325	3.26	4053	2.43	2.3
2	22371	3.32	11603	2.62	1.9
3	26864	4.65	12685	2.82	2.1
4	33928	6.05	15817	3.26	2.1
5	41148	6.57	17054	3.44	2.4
6	69439	8.21	28764	3.1	2.4
7	105674	15.3	66699	2.39	1.6
8	290561	18.5	84142	8.09	3.4

Table 1. Statistics of URLs and Clusters

We noticed the fact that the average links among URLs keep growing and there could be several reasons for this behavior. First and the most important, the crawler is conducting a breadth first traversal, and each time it starts at the same seed (www.uccs.edu), therefore it tends to first crawl the higher level pages, whose links are pointing to large number of pages yet to be crawled (Unknown dangling, see detail discussion in Section 5.2.2 The Rank Numbers in Each System) and can not be included

in the calculation. Later when the crawling limit increases, the target pages are retrieved and the links in previous pages are included. In future work, we recommend selecting the sample data in a better fashion to avoid this potential misleading factor, see detail discussion in Section 7.3.1 Better Sample Data. Second reason is that there are certain groups of pages in the UCCS domain with very large number of links (more than one hundred)[1], and they were simply not encountered at the earlier stages. The third reason is that probably we have not retrieved a good representation of the UCCS domain. When we stopped the crawler after data set 7 is retrieved, there are still more than 148,749 UCCS URLs in our database to be crawled, which is almost 1.5 times more than the total URLs in data set 7.

Except the last set, the average number of URLs per cluster remains relatively stable, although the sample data size has grown more than ten times during the entire experiment. This indicates that the outcome of the clustering algorithm is quite reliable.

Again in the first seven sets of data, the average links among clusters also remains relatively stable while the links among individual pages increases about five times. This behavior indicates that a very large group of links is actually "internal", meaning they only connect to pages within the same cluster.

The last set of data covers URLs from three universities, not just the UCCS domain, so it shows some different characteristics. As the dataset grows and the graph

---

[1] http://tle2.uccs.edu/webcal/month.php

gets closer to the real Web, the ratio between internal links and external links should remain stable according to a previous research [Bharat 2001] and this needs to be verified when larger amount of data become available.

5.2 Result Comparison

5.2.1 Top ranked URLs in each system

The most important questions one may ask is --- how does the new algorithm perform, and does it actually give high scores to the "most important" URLs? While the importance of a page is a rather subjective opinion, the original Page Rank algorithm gives quantitative measurement to it for us to use as a baseline.

Observations from other people have told us the first twenty hits in the results are the most important [Gordon 1999][Hawking 2001]. Therefore using the similar idea as a previous research [Gupta 2003], we look at the top 20 pages out of the 105,674 URLs according to Page Rank shown in Table 2. Top 20 pages in Original Page Rank (Crawled in September, 2006).

We immediately noticed that six spots are occupied by awfully similar URLs (#1 and #11 to #15). It is also quite surprising that the number #1 is actually not the root page of UCCS domain (www.uccs.edu). Further investigation shows that this page, as its URL implies, dynamically links to all press releases of UCCS in the past seven years. From each child page, there is a link back to the master list, resulted in an extremely high rank and also helped five of its children (#11 to #15) make to the top 20.

1	www.uccs.edu/~ur/media/pressreleases/article_list.php
2	www.uccs.edu
3	web.uccs.edu/ur/pr.htm
4	www.uccs.edu/campusinfo/campusinfo_form.htm
5	www.uccs.edu/directory/directoryb.htm
6	tle2.uccs.edu/webcal/month.php
7	www.uccs.edu/~webdept/approved
8	www.uccs.edu/%7Ewebdept/az/azstatic/A.html
9	web.uccs.edu/ur/topnav/topnav.htm
10	web.uccs.edu/library
11	www.uccs.edu/~ur/media/pressreleases/view_article.php?article_id=357
12	www.uccs.edu/~ur/media/pressreleases/view_article.php?article_id=358
13	www.uccs.edu/~ur/media/pressreleases/view_article.php?article_id=359
14	www.uccs.edu/~ur/media/pressreleases/view_article.php?article_id=356
15	www.uccs.edu/~ur/media/pressreleases/view_article.php?article_id=354
16	web.uccs.edu/library/library%20A_Z.htm
17	web.uccs.edu/library/databases/intro.htm
18	library.uccs.edu
19	web.uccs.edu/library/Library%20Information%20and%20Services/staffdir.htm
20	web.uccs.edu/library/remote.htm

Table 2. Top 20 pages in Original Page Rank (Crawled in September, 2006)

This fact certainly does not imply that the creator of the #1 URL wants to promote the URL over the UCCS root page. As discussed earlier in this report in section 3.3.4 The Impact of Adding New Pages, this is exactly the way how Page Rank should work and the behavior is in line with our prediction.

Let's look at the top 20's from Cluster Rank shown in Table 3. Top 20 pages in Cluster Rank (Crawled in September, 2006).

The top position is correctly taken by the root page of UCCS domain and the previous #1 is removed from the top 20, because in the new algorithm, internal links do not help the rank. The author personally considers this is a more desirable way to rank pages; however this question is certainly up for debate for the readers and other computer scientists.

New Rank	Old Rank	URL
1	2	www.uccs.edu
2	3	web.uccs.edu/ur/pr.htm
3	4	www.uccs.edu/campusinfo/campusinfo_form.htm
4	5	www.uccs.edu/directory/directoryb.htm
5	6	tle2.uccs.edu/webcal/month.php
6	10	web.uccs.edu/library
7	-	web.uccs.edu/wbahn/ECE1021/index.html
8	17	web.uccs.edu/library/databases/intro.htm
9	18	library.uccs.edu
10	19	web.uccs.edu/library/Library%20Information%20and%20Services/staffdir.htm
11	20	web.uccs.edu/library/remote.htm
12	7	www.uccs.edu/~webdept/approved
13	19	web.uccs.edu/library/Library%20Information%20and%20Services/hours.htm
14	16	web.uccs.edu/library/library%20A_Z.htm
15	9	web.uccs.edu/ur/topnav/topnav.htm
16	-	www.uccs.edu/%7Ewebdept/az/AZdir.php
17	-	www.colorado.edu
18	8	www.uccs.edu/%7Ewebdept/az/azstatic/A.html
19	-	web.uccs.edu/rlorch/default.htm
20		library.uccs.edu/search

Table 3. Top 20 pages in Cluster Rank (Crawled in September, 2006)

Fifteen out of twenty pages are also in the previous list, which is an accuracy of 75%, and the top 10 pages achieved accuracy of 90%. This is a satisfying ratio giving the simple and intuitive nature of the Cluster Rank algorithm, especially considering the fact that the previous list has six similar pages out of twenty.

However, an important note must be made that it is impossible for us to claim the Cluster Rank is “better” than the original Page Rank just based on the naïve comparison above. In fact, due to our limited amount of sample data, it is too early to jump into any conclusion.

5.2.2 The Rank Numbers in Each System

We noticed a fact that the rank numbers in the new system are all significantly lower than in the original. There are two explanations for this difference.

First, there are much less total points to work with in the new system. For example if there are 100 pages to rank, the original algorithm gives initial rank of 1 to all pages, which means there are total 100 points. In the new algorithm, if the 100 pages are grouped into 50 clusters, and we only calculate the rank at cluster level, obviously there are only 50 total points.

Second, the original algorithm must remove all dangling pages as they may play catastrophic roles in the calculation. See details earlier in this report. Section 2.1 The Original Page Rank algorithm. This factor can have big impact as it affects the total calculation space being dealt with. We noticed there are several kinds of dangling links:

- True dangling: The page is in fact not pointing to anywhere. This is also true when the URL is a binary file such as PDF or DOC.
- Hidden dangling: The page has complicated scripts to generate links to other pages, but the crawler can not figure out.
- Unknown dangling: The page has not been crawled yet, there the system simply does not know where does it link to, if any at all.

These problem children, however, can be included in the new algorithm provided they belong to a cluster that has other members connecting outside, which means as long as the whole cluster is not dangling, every member can be included. In our sample data, only 28488 URLs out of the total 105674 can be calculated in Page Rank system because of the dangling, but 66699 URLs are processed in Cluster Rank.

The combined effect of the two factors is that there are more members to share a smaller pie, and no doubt everybody gets a much smaller piece.

Further complications are introduced by congregating the links to cluster level. In this step, the links from individual links are merged together and could significantly change the structure of the graph as we already have seen earlier in this report. Section 3.3.4 The Impact of Adding New Pages.

While mathematical proof is rather difficult as the behavior of Page Rank itself also changes depending on the graph structure [Arasu 2001] [Kamvar 2003], we looked at the experimental data trying to identify a reasonable relationship:

1. If the cluster size is 1, meaning a page does not belong to any group and just stand alone by itself, the rank number in these two systems are relatively the same. We checked total of 8873 pages in this category, and find out the average ratio of between the rank numbers is 1.09, with standard deviation of 0.11.

2. For members of a tiny cluster, which has 2 to 10 URLs, its Page Rank could be expressed as the new cluster Rank times the size of the clusters it belongs to: $PR = CR * ClusterSize$. We checked the 3289 Pages in this category, and the average ratio of between the two sides of this equation is 1.14, with standard deviation of 0.35. However at this point the deviation is almost too large for us to make this statement. We can also consider the two scenarios are the same (first one has Cluster Size of 1), however the

difference in standard deviation indicates they are very likely differ from each other.

3. The above two scenarios accounted for 43% of the 28488 URLs we can compare. For other pages that do not belong to above two cases, we cannot recognize any pattern that can be supported mathematically.

In order to make a convictive statement, we suggest future the research to compare the entire rank matrix as well as the relative ratio among URLs in the two systems. A much more advanced mathematical model, such as Hidden Markov Model could be made to compare the transition probability between the two systems.

5.3 Performance Comparison

The Page Rank is implemented strictly following the pseudo code presented in Section 2.2 Page Rank Calculation Pseudo Code and Complexity without any optimization and tweaking. The Cluster Rank implementation follows the pseudo code in Section 3.3.5 The Complexity Analysis. Seven rounds of comparison were made. At each round calculation was performed three times and the average time was taken to minimize the impact of routine system activities that might be taking place.

The results are shown in Table 4. Seven rounds of comparison between Page Rank and Cluster Rank.

Total number of URLs	Page Rank time (s)	Cluster Rank time (s)	Difference
9325	386.5	340.2	12%
22371	840.4	916.6	-9%
26864	1305.9	1130.5	13%
33928	1968.1	1564.5	20%
41148	2195.8	1722.6	21%
69439	4455.6	3251.1	27%
105674	15914.5	10477.6	34%
290561	31246.1	26562.54	15%

Table 4. Seven rounds of comparison between Page Rank and Cluster Rank

Figure 8 can help us to see the performance data visually.

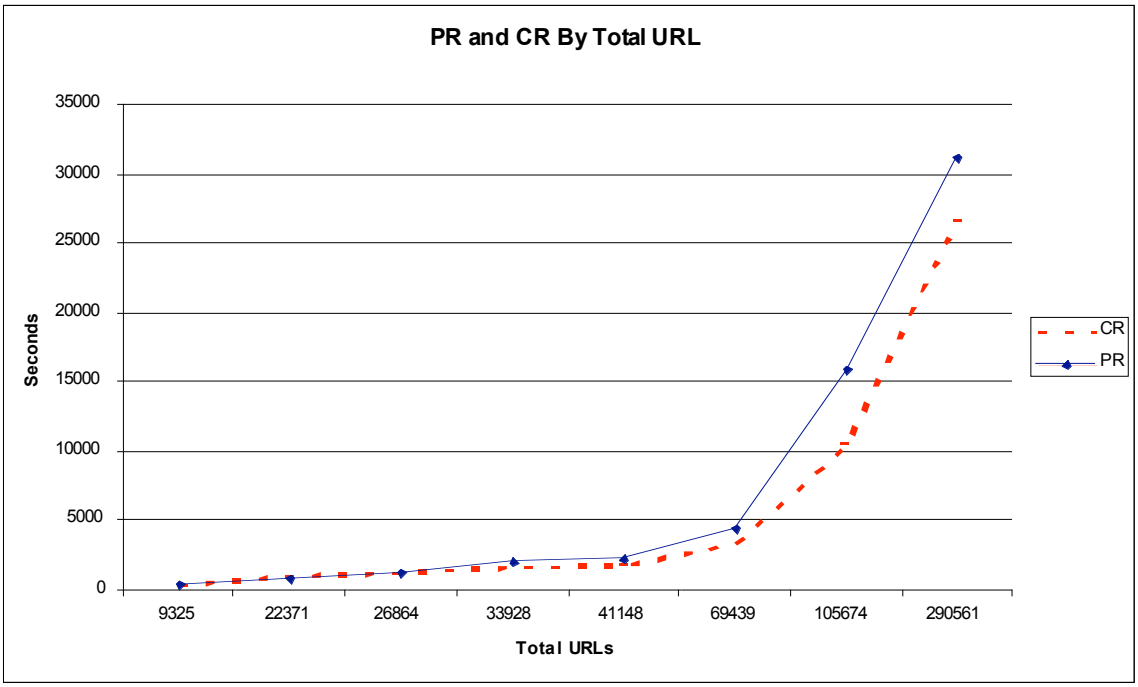


Figure 8: The performance chart between Page Rank and Cluster Rank

The percentage differences at each data point are illustrated in Figure 9. Percentage Difference.

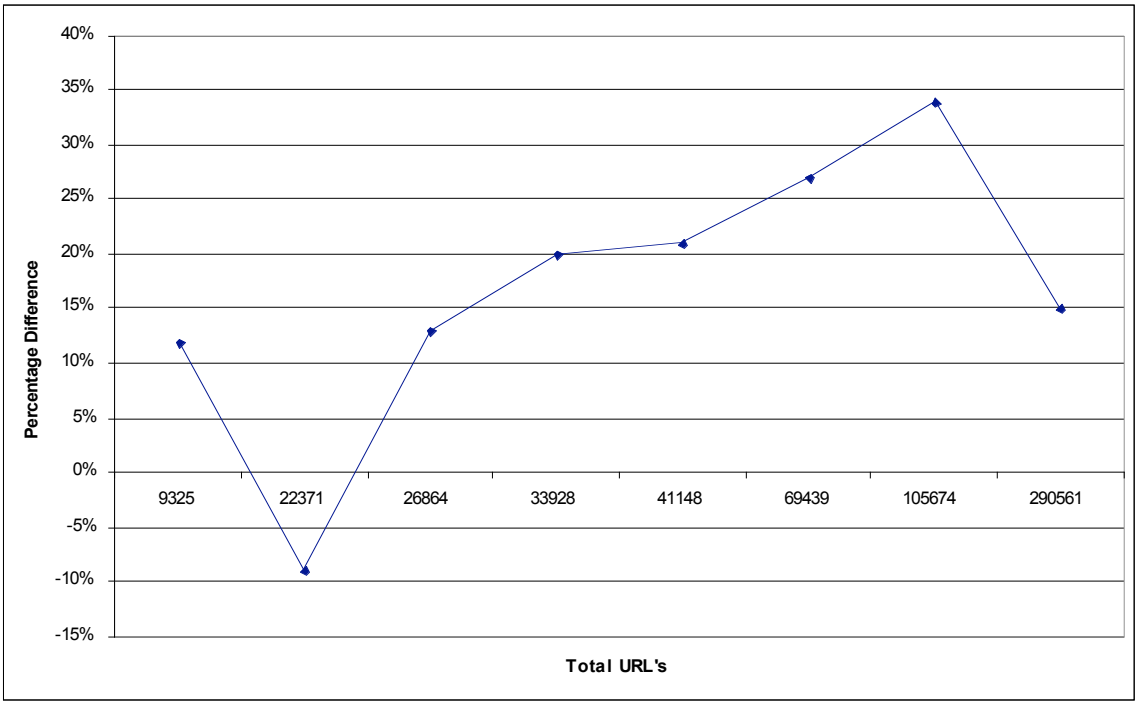


Figure 9. Percentage Difference

In most of the cases, Cluster Rank is 12% to 34% faster, with the average of 17% and standard deviation of 0.13. It falls behind of Page Rank at the second data point, which is very likely an unusual case and can be safely discarded.

5.4 Cluster Rank Calculation Break Down

The Cluster Rank calculation consists of several steps: first and second level clustering, data preparation to congregated links among URLs to the clusters and the Cluster Rank calculation itself. The final distribution step is not included in calculation as the weighted average computing is extremely fast and only counts a very small fraction of the total time. The break down table is as following (all numbers are seconds):

Total URLs	First Level Clustering	Second Level Clustering	CR Preparation	Cluster Rank	Total
9325	18.77	7.27	4.91	309.27	340.22
22371	112.13	30.76	10.70	763.04	916.63
26864	167.77	38.20	12.50	912.06	1130.53
33928	257.57	56.64	15.57	1234.76	1564.54
41148	283.11	69.60	16.41	1353.47	1722.59
69439	778.99	165.96	28.95	2277.20	3251.10
105674	3784.86	1142.76	65.42	5484.56	10477.60
290561	10706.70	3028.271	180.26	12647.309	26562.54

Table 5. Cluster Rank time break down for each step

The break down chart below clearly shows each step increases at a constant pace, in line with the observations in Section 3.3.5 The Complexity Analysis.

5.5 Choice of Density Threshold

The clustering threshold *t* must be carefully chosen as the distinction between sparse and dense graphs is rather vague. While choosing the density threshold for second level clustering, we want to consider two factors: performance and quality.

If the threshold is too high, almost no proposed clusters can be approved and the whole second level clustering would be meaningless. On the other hand if the threshold is too low, too many unrelated URLs may be grouped together and generate undesirable results.

The threshold selection process is conducted on a set of 138,430 URLs in UCCS domain. The density distribution of clusters on different rank is show below:

Graph Density	Number of Clusters in this range
0.9 ~ 1	5
0.8 ~ 0.9	38
0.7 ~ 0.8	36
0.6 ~ 0.7	57
0.5 ~ 0.6	97
0.4 ~ 0.5	57
0.3 ~ 0.4	65
0.2 ~ 0.3	326
0.1 ~ 0.2	423
0 ~ 0.1	479

Table 6. The distribution of Clusters on different density range

The data can be drawn to the following chart to help identify the characteristics:

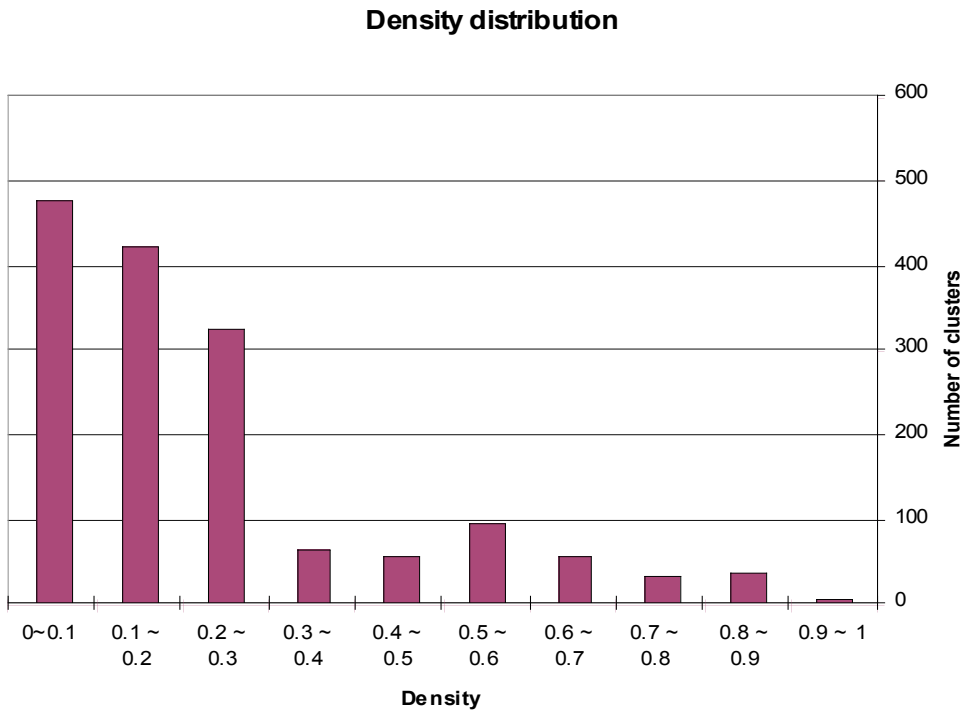


Figure 10, Density distribution of the proposed second level clusters

Naturally we see the big difference from the third group to forth group, which lead us to think threshold of 0.3 is potentially a good choice.

5.5.1 Performance

Experiments are performed at four different potential thresholds, 0.2, 0.3 0.5 and 0.7. The performance data, as expected, also show a sudden change at threshold of 0.3.

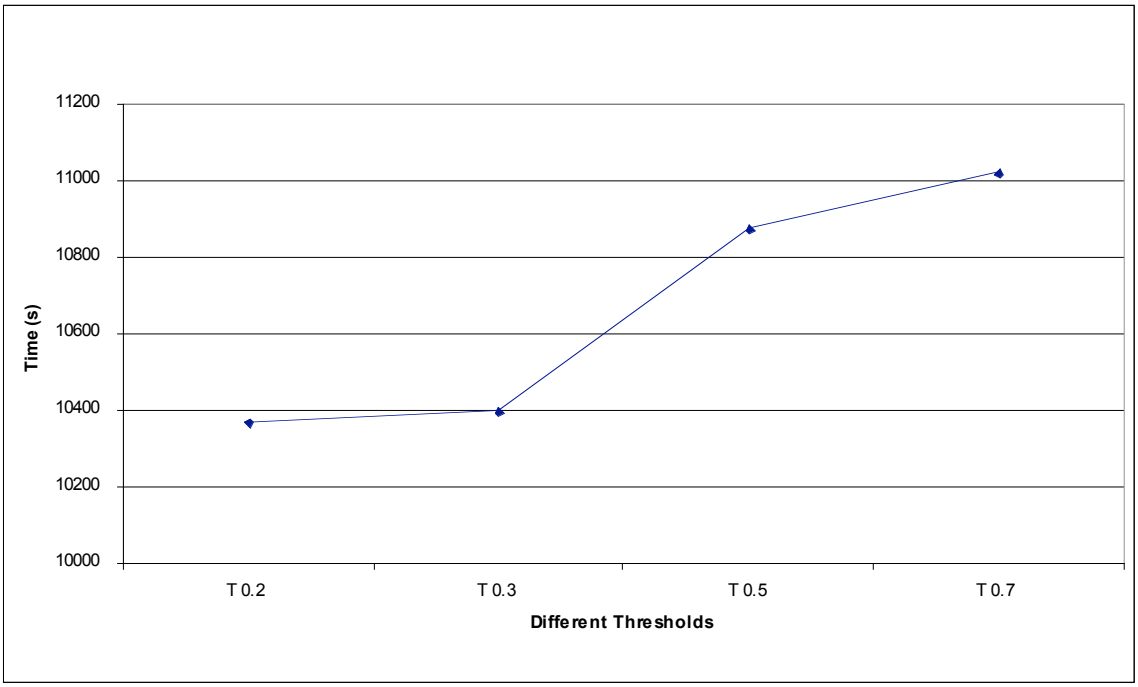


Figure 11. The Performance of second level clustering at different thresholds

5.5.2 Cluster Quality

Measuring the cluster quality is very important especially for the purpose of this algorithm, as we want to know if the clusters generated are truly representing the concept of "similar pages" and if it is actually fair to group them together.

Using cosine value between two vectors to measure the similarities between documents has been a well established mechanism and has been one of the most commonly used [Zamir, 1997] [Steinbach, 2000].

The basic idea is to break the document into distinct words, and then represent the whole document with a vector that consists of the frequency of each word. Obviously each document has its own vector. By computing the cosine between two vectors that representing the two documents, we comfortably say this value is the similarity between them and get a quantitative representation rather than casual human observation. When two documents are completely identical, the cosine value is 1, and the smaller it is, the more differences exist.

Implementation of above work can be a quite challenging task as it involves intensive computing and a process calling "stemming" to distinguish different format of the same word such as "like" and "likes", or "run" and "ran". Fortunately we are able to found a very good Perl module available to public[1] to conduct the experiments on our data.

For each cluster, a base URL is chosen and all other URLs are compared against it to compute the cosine value using the public available module. The averages of all cosine values become the quality of this cluster. We then compute the quality for each cluster in a given range. The average of all qualities and the standard deviation can objectively tell us the common characteristics of this range, which is what we need to decide where the threshold is, and it gives the tangible understanding of cluster quality.

As in the previous sections, the experiments were performed on five ranges, 0.2, 0.3, 0.5, 0.7 and 0.9. The results are as following:

---

[1] http://www.perl.com/pub/a/2003/02/19/engine.html?page=1

Density Range	Quality average	Standard Deviation
0.2 ~ 0.3	0.4	0.311
0.3 ~ 0.4	0.66	0.238
0.5 ~ 0.7	0.77	0.243
0.7 ~ 0.8	0.74	0.233
0.9 ~ 1	0.95	0.011

Table 7. The Average Quality of Clusters in Each Density Range.

The chart below can give better visual representation.

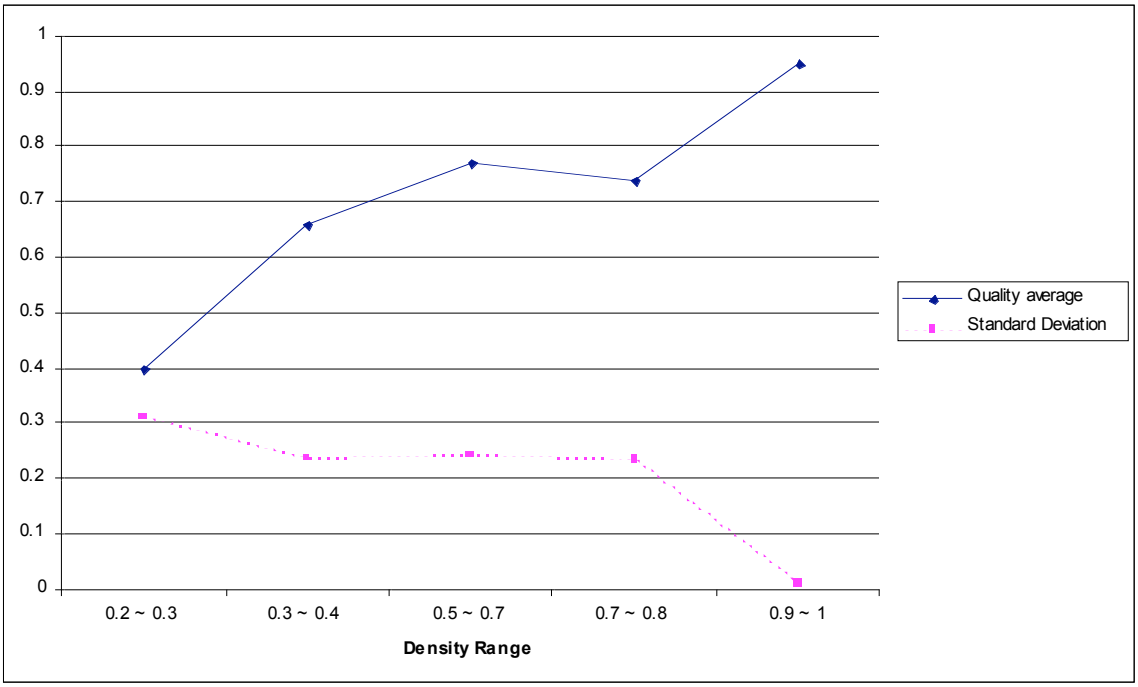


Figure 12. The Quality Trend and Standard Deviation

In this context, we see the higher density a cluster has, the higher quality (similarity) it also shows. The standard deviation is showing the inversed trend, which implies the clusters in the lower density range start to behave too differently. At density of 0.9 and above, all cluster members are almost identical. Ideally we would like to see all clusters behave this way, but there are only five clusters in this range from our sample data.

Of all the clusters at density of 0.3 and above, the cosine value (similarity) is above 0.5, which is a good choice as threshold following a previous research [Mobasher 1999]. Also, the sudden change happens at density of 0.3, implying the two groups are dramatically different, and also confirmed all the assumptions earlier. We also tested the similarity among 100 randomly selected pages from several definitely different groups, such as pages from Engineering[1], Business[2] and CNN website[3]. The average similarity is 0.29, with standard deviation of 0.11, which also strengthened the fact that similarity 0.5 is a comfortable choice.

To confirm the theory, we performed the following simple experiment on Yahoo[4] and Google[5]. Using the site specific search feature available from the two engines, we performed several simple searches on the UCCS domain. In their result lists, we randomly picked five clusters from each engine and calculated the similarity of the top 10 pages using the same program. For Yahoo, the average similarity is 0.53, with standard deviation of 0.16; and for Google, the average similarity is 0.48, with standard deviation of 0.08. We noticed that the "similar page" feature in Google can not be limited in one site only (i.e. cannot use site specific search and similar page search together). Its results have pages from various sites and tend to be less similar.

Adding all facts together, we finally chose graph density of 0.3 the threshold for the clustering algorithm presented in this report. All clusters with density of 0.3 and

---

[1] http://eas.uccs.edu/
[2] http://business.uccs.edu/
[3] http://www.cnn.com
[4] http://www.yahoo.com
[5] http://www.google.com

above have similarity (cosine distance between vectors) of 0.5 or higher, therefore they are considered closely related. Lower density clusters are discarded as they may have been put in the same virtual folder for other reasons.

5.6 Subjective evaluation

After all the computation and processing, when it comes to evaluate how "good" a search engine is, there is still nothing better than human eyes. The empty rank numbers probably do not mean much to the end users if they fail to find at least several important hits on the first or second page of the search results. In this experiment we gather a group of 10 people and ask them to answer the ten questions listed in Appendix A. The final results are listed below. The scores are on the scale of 1 to 5, 1 being the best. Standard deviation is given in parenthesis. Please see the result in Table 8. Subjective comparison from ten users.

	Yahoo	**Google**	**Needle(CR)**	**Needle (PR)**
Response time (seconds)	0.65 (0.03)	0.26 (0.02)	0.94 (0.1)	0.94 (0.07)
# of Results	100 ~ 140,000	100 ~ 210,00	8 ~ 3,000	8 ~ 8,000
First Page Accuracy	1.5 (0.52)	1.8 (0.42)	1.9 (0.31)	1.8 (0.4)
Second Page Accuracy	2.9 (1.3)	1.1 (0.31)	2.6 (0.84)	2.6 (0.84)
Order on the first page	1.3 (0.67)	1.5 (0.52)	1.6 (0.52)	1.6 (0.7)
Order on the second page	3.4 (0.7)	1.4 (0.52)	2.1 (0.31)	2.1 (0.56)
Related pages	10 ~ 87, 000	10 ~ 50	2 ~ 30,000	N/A
Related Pages Accuracy	1.5 (0.7)	4.5 (0.52)	1.7 (0.48)	N/A
Overall order of importance	1 (0)	1 (0)	1.5 (0.7)	1.5 (0.7)
Overall irrelevant hits	26% (0.08)	10% (0.04)	21% (0.08)	21% (0.09)

Table 8. Subjective comparison from ten users

It is not Needle's goal to compete with commercial giants. Based on this primitive and subjective evaluation, the Needle search engine performs relatively at the same level

as Yahoo or Google on the UCCS domain, giving the fact it has much smaller problem space to deal with, and the fact it is designed and tested on UCCS domain data from the beginning.

More importantly, we observed the two versions of Needle, which run on the two different algorithms (Page Rank and Cluster Rank) respectively, perform at the same level, which proved the value and design goal of Cluster Rank --- to provide reasonably results and the additional feature without performance loss.

CHAPTER 6

LESSONS LEARNED AND PROBLEMS SOLVED

When started to work on Needle project ten month ago, I only knew a handful of empty terms and several simple facts about search engines. As an computer professional, I have been using various search engines everyday but never thought about how they work and how much efforts their creators have put together to make them work up to people's expectations. In fact, quite often I got mad at search engines because they did not give me the correct answer in the first couple of result pages.

A search engine tries to parse and filter the entire Web, and automatically answer any questions that may be brought up by anonymous users. This has proven to be an extremely difficult job due to the sheer volume of data and the infinite number of possibilities and variations.

Working the Needle project, I have understood an incredible amount of details regarding search engines, from the architecture to all sub-components. Everything I have learned in Computer Science, especially Automata, Compiler, Database design, and Algorithm analysis, has contributed to the design and implementation process. In fact I quickly found myself exhausted and there was simply not enough background knowledge to support my research. Thanks to the guidance from Dr Kalita and the committee members, I was able to overcome many difficulties and at the same time learn many valuable lessons to be discussed in detail below.

6.1 Architectural Issues

My original thought was to create a very thin layer of database handler to transfer messages between the real database and all other modules, thinking a level of abstraction from the database may help creating a cleaner architecture. The idea was explored and abandoned immediately for three reasons: performance, scalability and compatibility. Currently there are two modules in the Needle project that are extremely database intensive: ranking system and text parser. In future if many users start to use it, the front end may become the third one. The speed of database operations has proven to be a bottleneck in the ranking calculation already, if another layer is added, the performance will only get worse and it could be prone to errors. Considering future growth, it is almost impossible to predict what operation each module might need and if the project is designed to have a common database handler, the scalability and compatibility can quickly become a road block.

What I would recommend, however, is to create the middleware layer for individual modules only when absolutely necessary, such as the case to export the graph data to a more efficient format for calculation, or cache front end queries. The lesson here is not to follow standard architecture blindly, but to adjust it to fit the need of the individual project.

6.2 Algorithm Choices

The ranking system is the heaviest part in this project. After all, seeking improvements to an already extremely successful algorithm is not easy. Needless to say

scientists have already been proposing changes during the past eight years (see Section 2.3. Improvements Proposed to Page Rank).

Deciding the idea of clustering is not difficult, but the details after are quite challenging. First, I had no almost knowledge of graph theory and clustering algorithms. Just to start the learning curve took a great amount of time. Second, it was extremely frustrating to find out that traditional clustering algorithms did not perform well in this particular context, and the very idea of clustering did not look promising.

After I stumbled upon the URL based clustering mechanism, the next problem was how to prove the whole algorithm was worthy. While the end results seem reasonable, I still cannot offer a complete convincing mathematical proof, which is in urgent need should the work continues.

6.3 Database Optimization

It takes many years to become a database expert and I only want to mention two important facts I learned in Needle project.

The first is to choose the correct data type and pass as few data items as possible among different modules. In the implementation of the text parser, a list of keywords from each page must be generated and inserted into the database, and later the total weight must calculated based on the location they appear. The first implementation was to use the keyword as index and pass them back and forth. This was a terrible choice in terms of storage space and performance because the handling of strings in a database needs much more work than integers. Thanks to the suggestion from fellow students, I

switched to creating a dictionary table and only passing the keyword IDs. The performance almost doubled while storage space was reduced by half.

The importance of creating a set of indices can not be reiterated enough. There are three tables, URL, Cluster and URLLinkStructure, in the Needle database to store the URLs, clusters and the links among them. The clustering ranking system queries these tables intensively during the calculation. The first implementation did not have any indices on them and the performance was unbearable --- more than 6 hours to calculate the Page Rank of 2000 URLs. After two unique indices were created, the speed is improved by order of magnitude achieving the numbers today --- a little over 3 hours for 100,000 URLs.

CHAPTER 7

LIMITATIONS AND FUTURE WORKS

In the process of designing and implementing the Needle search engine as well as the Cluster Rank algorithm, we made exciting progress but also identified many limitations that cannot be overcome easily and many virgin areas that we would like to cover but simply do not have the bandwidth and knowledge to explore. The following is a list of areas to be improved and they are categorized under each basic component of a search engine.

7.1 Crawler

A primitive implementation was written at very early stage of the project to retrieve some data for other modules to work with. While functioning correctly, this version rather is plain in terms of features: it is single threaded and does not have retrying, repository refreshing, URL hashing, smart checking on dynamic URLs, smart recognizing on file types, and avoiding crawler traps, etc. Its speed is also quite questionable and can only retrieve about 2000 URLs per hour on a fast network in the UCCS lab. Improvements can be made to add the features above and improve its speed. Fortunately two UCCS graduate students are already working on this area.

7.2 Parsers

Same as the crawler, a simple functional text parser was written to glue the whole system together. It only parses certain selected areas of a document such as metadata,

title, anchor text, three levels of headers, and a short part at the beginning of each paragraph. A complete full text parser with satisfactory performance is in immediate need. Image processing is not currently implemented.

7.3 Ranking System

Cluster Rank is the most important adventure in this report. While innovative and producing reasonable results, the lack of rigid mathematical proof and analysis is the weakest point. We see three major areas that need improvements and more works discussed below.

7.3.1 Better Sample Data

The limited amount and the nature of sample data is a major roadblock in our experiments. In Section 5.1 Sample Data, we see the average links among URLs keep increasing, indicating certain immatureness of the sample data sets. There are several ways to improve the quality of the sample data:

- Start at a smaller sub-domain of the UCCS domain, such as the Engineering website[1] and completely crawl all pages in it to get the complete representation of this sub-domain.
- Instead of always starting at the root page of the UCCS domain, each time use a different random seed URL, and crawl the same number for URLs (for example 10,000) to better reflect the nature of the Internet.

[1] http://eas.uccs.edu/

- After a very large data set is retrieved (for example one million URLs), randomly select a smaller subset (for example 100,000) as the test sample. This is similar to the previous approach but not the same in that the random selected pages may or may not connect to each other.

We predict that upon conducting the experiments on a series of better sample data, the result will be much more convincing than what is presented in this report.

7.3.2 Detail Analysis and Result Validation

As discussed earlier, Page Rank can be viewed as the stationary probability distribution over pages induced by a random walk on the Web [Haveliwala 1999]. To quantitatively evaluate the relationship between Page Rank and Cluster Rank presented in this report, we recommend the future research to create a HMM (Hidden Markov Model) to analyze the transition probability among the pages.

An alternative approach is to implement an automated performance comparison mechanism to provide quantitative results, and compare against other commercial search engines [Li 2000b].

7.3.3 Other Ranking System

For the past eight years, the Web community has depended on one or another static ranking system such as Page Rank. Recently scientists have applied machine learning techniques to this area and developed a ranking machine learning algorithm, and

achieved higher accuracy [Richardson 2006]. Future work may follow this direction to get more exciting results.

7.4 Front-end

The current front-end is quite efficient and clean in terms of searching and displaying the result. Its behavior and performance, however, is unknown when significantly larger amount of data become available. Improvements may not be urgent but the following few areas can be investigated: caching of search result to speed up duplicated searches, better organizing of the result, and capturing user behavior for future research.

The ability to search multiple words and phrases is also commonly available in search engines but yet to be implemented in Needle.

7.5 Technology

MySQL is chosen to host the entire engine because it is the most common open source database. The implementation language is Perl because of its stellar ability to parse text strings, process regular expressions and rich set of modules for various purposes. In all other modules except ranking system, this choice proves to be viable and efficient.

However in the computational intensive work such as Page Rank calculation, these two technologies to get a little overwhelmed. This module has to make constant inquiries to the link information, and even after building indexes, the database querying

has always been a big bottleneck. Looking at our performance data in Section 5.3 Performance Comparison, it is far slower than the work done by other scientists [Kamvar 2003].

Recently scientists have developed a new algorithm to store web graphs by exploiting gaps and references at an extremely high compression ratio [Boldi 2005]. For instance, they can save and compress a snapshot of about 18,500,000 pages of the .uk domain at 2.22 bits per link, and traverse the entire graph in a matter of seconds.

In future work, it is strongly recommended to export and save link data to this system for rank calculation.

CHAPTER 8

CONCLUSION

The Needle project is a simple yet completely functional search engine currently running on University of Colorado Systems. It also has a flexible, scalable and extensible architecture for adding new modules as well as improving existing ones. At the time this report is written, many other people are already working on the improvements of various modules or very close to completion.

While developing this project, this report also presented a valuable adventure of Cluster Rank algorithm, which serves as an alternative of existing Google Page Rank. Promising result and an additional feature of “similar pages” can be generated with noticeable performance improvement.

Initial evaluation and comparison are also presented in this report, showing satisfying result. However, the definitive comparison between the two algorithms and the comprehensive analysis are yet to be made on advanced mathematical models.

Rising on the horizon of Internet, search engine will no doubt be the focus of study for years to come. The Needle project has contributed its share and will continue to serve as a solid foundation for future academic works in UCCS.

BIBLIOGRAPHY

[Arasu 2001] Arasu, Arvind, J. Novak, A. Tomkins and J. Tomlin. Page Rank Computation and the Structure of the Web: Experiments and Algorithms, Technical Report, IBM Almaden Research Center, Nov. 2001.

[Bharat 2001] Bharat, Krishna, Bay-Wei Chang and Monika Henzinger. Who Links to Whom: Mining Linkage betweenWeb Sites. In Proc. of the IEEE Intl. Conf. on Data Mining, pages 51–58, 2001.

[Boldi 2005] Boldi, Paolo and Sebastiano Vigna. Codes for the World−Wide Web. Internet Math., 2(4):405-427, 2005.

[Brin 1998] Brin, Sergey and Lawrence Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. Proceedings of the Seventh International Conference on World Wide Web 7, Brisbane, Australia, Pages 107 – 117, 1998.

[Chen 2002] Chen, Yen-Yu, Qingqing Gan, and Torsten Suel. I/O-efficient techniques for computing Page Rank, Technical Report, CIS Department, Polytechnic University, 11/08/2002.

[Chirita 2004] Chirita, Paul - Alexandru, D. Olmedilla, andW. Nejdl. Pros: A Personalized Ranking Platform for Web Search. Technical Report, L3S and University of Hannover, Germany, Feb 2004.

[Chu 1996] Chu H and M. Rosenthal. Search Engines for the World Wide Web: A Comparative Study and Evaluation Methodology. In the *ASIS'96*, 59th American Society for Information Science and Technology Annual Meeting, Medford, NJ, Information Today, Inc., Pages 127-135, 1996.

[Dean 1999] Dean, Jeffrey and M. R. Henzinger. Finding Related Pages in the World Wide Web. Computer Networks, Amsterdam, Netherlands, Volume 31, Numbers 11-16, Pages 1467–1479, 1999.

[Dongen. 2000] Dongen, S. van. A cluster algorithm for graphs. Technical Report INS-R0010, National Research Institute for Mathematics and Computer Science, Amsterdam, The Netherlands, May.

[Fox 1997] Fox, Armando, S. Gribble, Y. Chawathe, E. Brewer and P. Gauthier. Cluster-Based Scalable Network Services. In Proceedings of the SOSP'97, Symposium on Operating Systems Principles, St. Malo, France, 1997. http://www.cs.berkeley.edu/~brewer/papers/TACC-sosp.pdf

[Gilfillan 2001] Gilfillan, Ian. Optimizing MySQL: Queries and Indexes. Visited November, 2006 http://www.databasejournal.com/features/mysql/article.php/1382791

[Gordon 1999] Gordon, Michael and Praveen Pathak. Finding information on the World Wide Web: the retrieval effectiveness of search engines. Information Processing and Management, 35:141-180, 1999

[Guillaume 2002] Guillaume, Jean-Loup and Matthieu Latapy. The Web Graph: an Overview. LIAFA - Universit´e Paris 7, 2, place Jussieu, 75005 Paris, France.

[Gupta 2003] Gupta, Ankur and Sreangsu Acharyya. Cluster Enhanced Page Ranks. Technical Report. The University of Texas at Austin. December 15, 2003.

[Harpf 2005] Harpf, Lauri. Free website promotion tutorial. Visited Nov, 2005. http://www.apromotionguide.com/

[Hartigan 1979] J. A. Hartigan, M. A. Wong. A K-Means Clustering Algorithm. Applied Statistics, Vol. 28, No. 1 (1979), pp. 100-108.

[Haveliwala 1999] Haveliwala, Taher H. Effcient Computation of Page Rank. Technical Report. Stanford University, California, 1999 http://dbpubs.stanford.edu:8090/pub/1999-31

[Hawking 2001] Hawking, David and Nick Craswell. Measuring search engine quality. Information Retrieval, 4(1), 33–59,

[Heydon 1999] Allan Heydon, Marc Najork, Mercator: A scalable, extensible Web Crawler, World Wide Web 2, Pages 219-229, 1999.

[ht://dig 2002] The ht://Dig Group. 2002. Accessed November, 2005. http://www.htdig.org/

[Jacobs 2005] Jacobs, Jing. CatsSearch An Improved Search Engine Design For web pages in the UCCS Domain. University Of Colorado at Colorado Springs, December, 2005.

[Jansen 2000] Jansen, Bernard, Abby Goodrum and Amanda Spink. Searching for multimedia: analysis of audio, video and image Web queries. World Wide Web 3, Pages 249-254, 2000.

[Joachims 1997] Joachims, Thorsten, D. Freitag, and T. Mitchell. Webwatcher: A Tour Guide for The World Wide Web. In Proceedings of the IJCAI'97, Fifteenth International Joint Conference on Artificial Intelligence, Nagoya, Aichi, Japan, Pages 770–777, August 1997.

[Johnson 1967] Johnson, S.C. Hierarchical clustering schemes. Psychometrika, 32, pp. 241-253

[Kamvar 2003] Kamvar Sepandar D, Taher H. Haveliwala, Christopher D. Manning, and Gene H. Golub. Extrapolation methods for accelerating Page Rank computations. The 12th International. Conference on the World Wide Web, pages 261–270, 2003.

[Kim 2002] Kim, Sung Jin and Sang Ho Lee. An improved computation of the Page Rank algorithm. The European Conference on Information Retrieval (ECIR), pages 73–85, 2002.

[Kodavanti 2005] Kodavanti, Apparao. Implementation of an Image Search Engine. University Of Colorado at Colorado Springs, December, 2005.

[Lee 2003] Lee, Chris P., Gene H. Golub, and Stefanos A. Zenios. A fast two-stage algorithm for computing Page Rank. Technical report, Stanford University, 2003.

[Li 2000b] Li, Longzhuang and Yi Shang. A New Method For Automatic Performance Comparison of Search Engines. World Wide Web 3, Pages 241-247, 2000.

[Li 2000a] Li, Wen-Syan, Okan Kolak, Quoc Vu, and Hajime Takano. Defining Logical Domains in a Web Site. In Proceedings of the 11th ACM Conference on Hypertext, pages 123 - 132, San Antonio, TX, USA, May 2000.

[Lopresti 2000] Lopresti, Daniel and Jiangying Zhou. Locating and Recognizing Text in WWW Images. Information Retrieval 2, Pages 177-206, 2000.

[McCallum 2000] McCallum, Andrew, Kamal Nigam, Jason ï Rennie, and Kristie Seymore. Automating the onstruction of Internet portals with machine learning. Information Retrieval Journal, 3:127–163, 2000

[McSherry 2005] McSherry, Frank, A Uniform Approach to Accelerated Page Rank Computation, International World Wide Web Conference, Chiba, Japan, May 10-14, 2005.

[McSherry 2005] McSherry, Frank. A Uniform Approach to Accelerated PageRank Computation. International World Wide Web Conference, Chiba, Japan, May 10-14, 2005.

[Mobasher 1999] Mobasher, Bamshad, Robert Cooley and Jaideep Srivastava. Automatic personalization based on web usage mining. TR99-010, Department of Computer Science, Depaul University, 1999.

[Morgan 2001] Morgan, Eric. Comparing Open Source Indexers. O'Reilly Open Source Software Conference, San Diego, CA, July 23-27, 2001. http://www.infomotions.com/musings/opensource-indexers/

[Nowack 2005] Nowack, Craig. Using Topological Constructs To Model Interactive Information Retrieval Dialogue In The Context Of Belief, Desire, and Intention Theory. Dissertation of Ph.D. Pennsylvania State University, Pennsylvania, 2005. http://etda.libraries.psu.edu/theses/approved/WorldWideFiles/ETD-848/Dissertation.pdf

[Pandey 2005] Pandey, Sandeep and Christopher Olston, User-Centric Web Crawling, International World Wide Web Conference, Chiba, Japan, May 10-14, 2005.

[Pitkow 1994] Pitkow, James and Margaret M. Recker. A Simple Yet Robust Caching Algorithm Based on Dynamic Access Patterns. Proceedings of the Second World Wide Web Conference (WWW2), Chicago, IL, 1994.

[Rasmussen, 1992] E. Rasmussen. Clustering Algorithms. In W. B. Frakes and R. Baeza-Yates (eds.), Information Retrieval, pages 419-42. Prentice Hall, Eaglewood Cliffs, N. J., 1992.

[Richardson 2006] Richardson, Matthew, Amit Prakash and Eric Brill. Beyond PageRank: Machine Learning for Static Ranking. International World Wide Web Conference, Edinburgh, Scotland, May 23-26, 2006

[Salton 1983] Salton, Gerard, and Michael J. McGill. Introduction to Modern Information Retrieval. New York: McGraw-Hill Book Co. 1983.

[Smyth 1997] Smyth, P. Clustering Sequences Using Hidden Markov Models, in Advances in Neural Information Processing 9, M. C. Mozer, M. I. Jordan and T. Petsche (eds.), Cambridge, MA: MIT Press, 648{654, 1997.

[Steinbach 2000] Steinbach, M., George Karypis and Vipin Kumar, V. 2000. A comparison of document clustering techniques. 6th ACM SIGKDD, World Text Mining Conference, Boston, MA.

[Takano 2000] Takano, Hajime and Terry Winograd. Dynamic Bookmarks for the WWW. In Proceedings of the 1998 ACMHypertext Conference, pages 297–298.

[Wall 2005a] Wall, Aaron. History of Search Engines & Web History. Visited November, 2005.
http://www.search-marketing.info/search-engine-history/

[Wall 2005b] Wall, Aaron. Future of Search Engines. Visited November, 2005. http://www.search-marketing.info/future-of-search-engines/index.htm

[Wang 2004] Wang, Yuan and David J. DeWitt. Computing Page Rank in a distributed internet search system. In Proceedings of the 30th VLDB Conference, 2004.

[Zamir, 1997] Zamir, Oren, Oren Etzioni, Omid Madani and Richard M. Fast and Intuitive Clustering of Web Documents. In Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining, pages 287-290, 1997.

Appendix A Survey Questions to Evaluate Needle

Perform 20 different searches on Needle (Cluster Rank), Needle (Page Rank), Yahoo (site: uccs.edu) and Google (site: uccs.edu).

1. Response time
2. Number of results estimated
3. First Page accuracy (scale 1 to 5, 1 being the best)
4. Second page accuracy (scale 1 to 5, 1 being the best)
5. Result Order on the first page (scale 1 to 5, 1 being the best)
6. Result Order on the second page (scale 1 to 5, 1 being the best)
7. Click the “similar pages” or “more from this site”, how many hits are returned?
8. Click the “similar pages” or “more from this site”, and check the first 10 to 20 hits, are they truly “related” to the original hit? (scale 1 to 5, 1 being the best)
9. Overall, are the important pages showing up early? (scale 1 to 5, 1 being the best)
10. Overall, the percentage in result hits are irrelevant? (Give a percentage)

Appendix B Software and hardware environment

Hardware environment:

Pentium 4, 2.0G CPU and 1G RAM.

Software environment:

Fedora Core 4, MySQL server 5.0.26, Perl v5.8.6. Apache/2.2.3

Appendix C Database Scripts

Creation

```
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,
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),
PRIMARY KEY (url_id) );

CREATE TABLE PageRank (
url_id          INT UNSIGNED NOT NULL,
out_link_count  INT UNSIGNED,
c_date          DATE,
c_pr            FLOAT ZEROFILL,
old_date1       DATE,
old_pr1         FLOAT ZEROFILL,
old_date2       DATE,
old_pr2         FLOAT ZEROFILL,
cal_current_iter    SMALLINT UNSIGNED,
UNIQUE INDEX (url_id),
CONSTRAINT FOREIGN KEY(url_id) REFERENCES URL(url_id)
ON DELETE CASCADE ON UPDATE CASCADE
);

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),
cluster_size        INT UNSIGNED,
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 TABLE SearchHistory (
ip_address      VARCHAR(16) NOT NULL,
host_name       VARCHAR(255),
request_time    DATE,
search_words    VARCHAR(255),
page_number     INT UNSIGNED
);



CREATE OR REPLACE VIEW vwPRCRComparison
AS
SELECT PageRank.url_id, URL.cluster_id, URL.url, PageRank.c_pr,
PageRankByCluster.c_prc, PageRank.c_pr/PageRankByCluster.c_prc ratio
FROM PageRank, PageRankByCluster, URL
WHERE PageRank.url_id = PageRankByCluster.url_id and PageRank.url_id =
URL.url_id
ORDER BY PageRank.url_id;

CREATE OR REPLACE VIEW vwURLCountInCluster
AS
SELECT COUNT(url_id) size, cluster_id FROM
URL
where cluster_id is not null GROUP BY cluster_id;
```

Cleaning

```
DROP VIEW vwPRCRComparison;
DROP VIEW vwURLLinkCluster;
DROP TABLE MediaType ;
DROP TABLE URL;
DROP TABLE Crawler;
DROP TABLE ImageProcessor;
DROP TABLE URLLinkStructure;
DROP TABLE Page Rank;
DROP TABLE PageLocation;
DROP TABLE TextParser;
DROP TABLE KeyWord;
DROP TABLE KeyWordWork;
DROP TABLE Dictionary;
DROP TABLE Page RankByCluster;
DROP TABLE SecondLvlClusterWork;
DROP TABLE Cluster;
DROP TABLE SearchHistory;
commit;
```

Appendix D Installing the required Perl Modules

Several Perl modules being used in Needle program may not be available in a standard Perl installation:

- Bundle::XML
- Tie::IxHash
- Time::HiRes
- HTML::TokeParser

These modules can be installed by using “cpan” command while logging in as the root user.

Appendix E Execution Steps

1. Configuration:

   Almost every aspect of the Needle search engine is controlled by the Config.xml, which has six major sections: DB, crawler, textparser, ImageProcessing, RankingSystem and FrontEnd. All settings are commented with notes to explain their purposes.

2. Crawling:

   perl unicrawler.pl

3. Ranking (may choose either one, or both):

   a. Original Page Rank: perl Page Rank.pl

   b. Cluster Rank:

      perl clustering.pl

      perl clusterrank.pl

4. Text parsting:

   perl textparser.pl

Appendix F Using the search engine

1. Enable Perl CGI configuration for the Apache web server.

2. To search the Cluster rank result, use:

   http://<hostname > /cgi-bin/search.pl

3. To search the Original Page Rank, use:

   http://<hostname > /cgi-bin/oldsearch.pl