

Improving Authoritative Sources in a  
Hyperlinked Environment via  
Similarity Weighting  
Or, “How to get better search results on the Web”

Jonathan D. Herbach  
Department of Computer Science  
Princeton University  
jherbach@cs.princeton.edu

May 2001

A Computer Science BSE Thesis

Prof. Andrea S. LaPaugh, *Advisor*

This paper is my own work in accordance with University Regulations.

*Jonathan Herbach*

## **Abstract**

Recent literature demonstrates that the network structure of a hyperlink environment can serve as an effective source for inferring the importance of content in documents. One such connectivity-analysis algorithm, HITS, determines document importance based upon the hyperlink structure of the Web. In order to compensate for the problems of a pure connectivity-analysis algorithm, we develop and test an algorithm based upon HITS that also considers document content. We investigate whether incorporating the similarity of documents joined by links modifies how the importance of documents is determined. While using similarity does not always compensate for existing problems in pure connectivity-analysis algorithms, similarity provides additional insight into why certain problems exist. The results also demonstrate that importance in our algorithm is often biased by the structure and layout of documents.

## Acknowledgments

This thesis, and the HITS-SW research that accompanied it, would not have been possible without the help of many other people.

Though it started as a four month project, my advisor, Prof. Andrea LaPaugh, happily guided my research for two semesters. Whether conversations were on opposite ends of campus or opposite coasts, she was consistently able to translate my instincts and ideas into concrete concepts.

Dr. Kevin Wayne and Prof. LaPaugh both served as official readers and sounding boards for ideas of this research. I regret not having the time to follow through on all their suggestions.

I am grateful to all who read various drafts of this work. Two people in particular continually revisited this engineer's prose: Marah Katz and Andrew Sobel. Without their help, this document would consist solely of ASCII art.

James Roberts of the Computer Science Department and James Chu at CIT obtained the computational resources necessary to complete this research. Dwight Bashore and Thomas Knowles extracted URLs from the Princeton and Computer Science search engines to provide a basis for my implementation.

Much of my work was created high above land and often far from Princeton. Such endeavors would not have been possible without laptop computers provided by Ike Himowitz, Bruce Hunt, David Kulansky, and Rita Saltz.

My friends  $\in$  202 Scully Hall and 2820 Alhambra Drive were willing to put up with my odd hours. I appreciate the freedom they they afforded me.

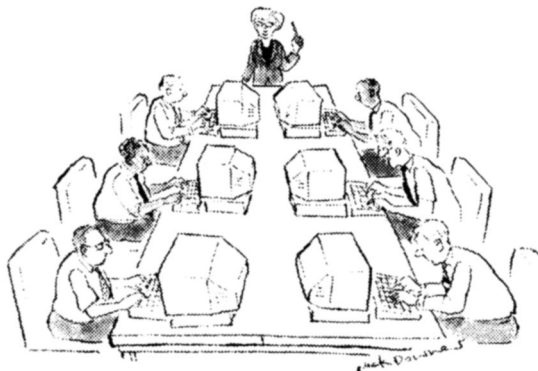
Finally, I am grateful to my parents, Etta and Mark Herbach, for their support provided throughout this process of programming, analyzing, writing, and editing. I appreciate all they have done to enable me to finish this work.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Guide to the Literature</b>	
	<b>Or, How does HITS-SW fit into a search engine?</b>	<b>3</b>
2.1	User Interface . . . . .	3
2.2	Gathering Documents for a Collection . . . . .	3
2.3	Searching Through a Collection for Related Documents . . . . .	4
2.4	Ranking Documents According to Relevance . . . . .	5
2.4.1	The HITS and PageRank algorithms . . . . .	5
2.4.2	Additional developments in structure-based ranking . . . . .	9
2.5	Performance and Architecture . . . . .	10
2.6	Search Engine Quality . . . . .	10
<b>3</b>	<b>Information Access, Storage and Retrieval</b>	
	<b>Or, What does “similar” mean?</b>	<b>11</b>
3.1	Indexing . . . . .	11
3.2	Similarity . . . . .	11
3.3	Term Vectors . . . . .	12
3.3.1	Refining of term vectors . . . . .	13
3.3.2	Comparing term vectors . . . . .	13
<b>4</b>	<b>Experiment Design</b>	
	<b>Or, What is HITS-SW?</b>	<b>15</b>
4.1	Policies . . . . .	17
4.1.1	Gathering Documents . . . . .	17
4.1.2	Locating Relevant Documents . . . . .	18
4.1.3	Ranking Documents Based Upon Relevance . . . . .	18
4.2	Similarity . . . . .	18
4.2.1	Binary Term Vector Approach . . . . .	18
4.2.2	Simplified Similarity Metrics . . . . .	19
4.2.3	Candidates for Similarity Weights . . . . .	19
4.2.4	Final Similarity Weights . . . . .	23
4.3	Obtaining and Measuring HITS-SW Results . . . . .	24
4.3.1	Testing HITS-SW . . . . .	24
4.3.2	Measuring Results . . . . .	24
4.3.3	Creating a control group with random similarity weights . . . . .	25
<b>5</b>	<b>Implementation</b>	
	<b>Or, How do we collect 8 GB of data?</b>	<b>27</b>
5.1	Database Schema . . . . .	27
5.2	Seeding the Database . . . . .	28
5.3	Core Library Functionality and Performance Designs . . . . .	30
5.4	Database Information . . . . .	31
5.5	Performance Concerns . . . . .	31
5.6	Gathering Data from the Web . . . . .	31
5.7	Assigning Similarity Weights to Hyperlinks . . . . .	32

5.8	Preparing a Query-Specific Collection for HITS-SW . . . . .	33
5.9	Hubs and Authorities Computation . . . . .	33
5.10	Manipulating Ranked Output . . . . .	34
5.11	Random Weighting Function for Control Group Tests . . . . .	34
5.12	Security, Performance, and Political Concerns . . . . .	34
<b>6</b>	<b>HITS-SW Findings</b>	
	<b>Or, As a result, did the sharks get smarter?</b>	<b>35</b>
6.1	Randomized Control Groups Are Unlike HITS-SW Tests . . . . .	35
6.2	Example HITS-SW Queries . . . . .	39
6.2.1	“java” at Princeton . . . . .	39
6.2.2	“shrimp” at Computer Science . . . . .	40
6.2.3	“sex” at Princeton . . . . .	43
6.3	Analysis of HITS-SW Experiments . . . . .	43
6.3.1	HITS as similar to HITS-SW <i>lo</i> . . . . .	43
6.3.2	Clustering in HITS and HITS-SW . . . . .	46
6.3.3	Analysis of a small query . . . . .	47
6.3.4	Structural Similarity vs. Content Similarity . . . . .	47
6.3.5	Using Links for the Similarity Measurement in HITS-SW . . . . .	48
<b>7</b>	<b>Conclusions</b>	
	<b>Or, Where will you next see HITS-SW?</b>	<b>49</b>
<b>A</b>	<b>Aggregate Entropy Data</b>	
	<b>Or, Will the reader examine indistinguishable numbers?</b>	<b>53</b>

# 1 Introduction



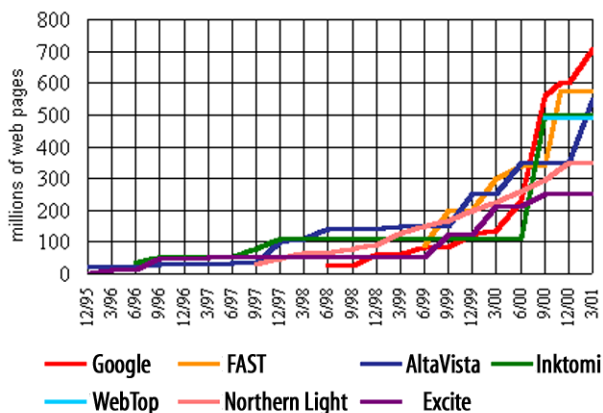
"Gentlemen, start your search engines."

[16]

The amount of information at our fingertips—on the Web, in libraries, in reference databases, etc.—is increasing every day at an enormous rate. In 1998, the Web was estimated at 800 million pages, composed of roughly 15 TB of actual content (with roughly 30% of this as text). [22] The size of the web has been increasing exponentially.

Search engines are an important development as a means to access such a large body of data. The number of pages that search engines index has grown as well. Figure 1 shows the recent growth in the quantity of search engine indexes.

Figure 1: Search Engine Index Sizes [6]



Google<sup>1</sup> is one of the more popular Web search engines, and with roughly 706 million pages indexed today, it has the largest index of the Web.

The size of an index is important, as it increases the probability that a user will find desired information. It is also important to consider how a search engine determines what information is important. If one searches the Web for “Internet”, one is bound to find many results. An algorithm will rank these results based upon the importance of the documents. One type of approach examines the content of the documents and determines importance accordingly; for example, importance could be determined by frequency of the

<sup>1</sup><http://www.google.com>

word “Internet”. Another approach uses the topology and connectivity of the documents to determine importance. The Hyperlink-Induced Topic Search (HITS) algorithm is one such connectivity-analysis algorithm. For pages on the Web, HITS infers document importance from hyperlinks, and ranks a set of pages accordingly. [20]

The research reported in this paper focuses on improving the HITS algorithm by including similarity information. Weights are assigned to all hyperlinks based upon the similarity of the source and destination pages. Similarity is determined using term vectors and other classic information retrieval techniques.

The HITS algorithm is modified to consider only a subset of the links as input. We call this new algorithm HITS-SW, as it incorporates similarity weighting into the original HITS algorithm. HITS-SW takes an additional parameter that describes the range of similarity weights that should be considered. (HITS is logically HITS-SW with the complete range of similarity weights considered.)

Dozens of queries have been ranked with HITS-SW using links weighted greater than (*hi*) and less than (*lo*) the median weight value. Each query has also ranked with HITS-SW using randomly assigned similarity weights to act as a control. Quantitative metrics developed to compare the differences in orderings show distinct differences between the randomly-assigned weights, *hi*, and *lo* HITS-SW rankings.

Conclusions include:

- The problem of document clustering in HITS still persists in HITS-SW.
- Many HITS rankings are most similar to *lo* HITS-SW rankings.
- HITS-SW uses similarity weights often focused on the similarity in structure and layout between documents and not actual content. Thus, pages produced from the same layout template will often be considered very similar even if the actual content varies.

## 2 Guide to the Literature

### Or, How does HITS-SW fit into a search engine?

There are several disparate issues related to an operating search engine. Fundamentally, they include providing an interface for the user to interact with a search service; designing algorithms to catalog, organize, and retrieve documents; architecting a system to scale with massive quantities of data; and considering how best to determine whether such a service provides good results.

#### 2.1 User Interface

From the user's perspective, a search engine like Yahoo<sup>2</sup> provides an intermediary, directory-like organization of data. Each document, categorized by humans, allows the user to browse documents much like he would in the shelves of a library. (Alternatively, the user can jump directly to a category, by entering a query for the category but the principle is the same.)

A search engine such as Google's primary interface is a "content-query" interface to the data. Users are allowed to enter phrases describing the content of the desired documents. Then the software determines which documents are relevant and the relative importance of each document.

While both styles allow the user to retrieve information, the interface design—an important part of an operating search engine—will not be the focus of this paper.

#### 2.2 Gathering Documents for a Collection

In libraries there is logically one means for adding more content to the *universe*—i.e., total collection. For example, the librarian who selects new books for a library has the responsibility of entering a record into the card catalog. In the current WWW model, there are many publishers of information. Each webmaster manages the pages for his domain, while each individual manages the data on his homepage. As such, there are many ways to add new information to the universe of content. Thus, while there is no centralized mechanism for loading data into search engines, the process is a controlled, albeit manual, process. Two approaches include:

1. Publishers can notify search engines of new content to be indexed.
2. Search engines can automatically seek out new content.

If the first approach were followed universally, search engines would contain solely those pages actively being promoted by publishers.<sup>3</sup> Alternatively, consistent application of the second approach would result in significant coverage gaps as many pages would escape notice.

Search engines act as *spiders* on the Web and collect information. This *crawling* process can be accomplished by depth- and breadth-first searches. Figure 2 provides a simple framework for a spider.

---

<sup>2</sup><http://www.yahoo.com>

<sup>3</sup>The cynical reader may claim that the current Web reflects this. Indeed, many search engines have a significant bias towards self-promoted pages, and this bias varies greatly between different search engines. (Are the promoted pages better? Perhaps.) Note that many companies provide search functionality purely from an advertising revenue model. Companies must weigh integrity of results with preferential treatment to those providing an income stream.

Figure 2: Pseudocode for a simple spider

```
For each known, but un-indexed document  $i$  {  
  Retrieve  $i$ , and store the content for future searching  
  For each link in  $i$  to other documents  $j_1 \dots j_n$  {  
    if  $j$  is an unindexed document {  
      add  $j$  to set of known documents  
    }  
  }  
  Mark document  $i$  as indexed  
}
```

As it becomes less feasible to construct a complete index (in terms of the number of pages indexed), the *quality* of an index becomes more important. Recent research into algorithms to approximate a complete index and quantify the quality of the approximation of an index are particularly useful with respect to new and more efficient methods of crawling. [19] Some of the current literature describes ways to crawl the Web in a more efficient manner. Crawling is a resource-intensive process, especially considering that a spider must recrawl content periodically. Although one could await improvements in the speed of computer hardware, it is better to pursue a different algorithmic approach to crawling as the growth of documents to index rivals that of the improvements in computer hardware.

The design of a good crawler presents many challenges. Externally, the crawler must avoid overloading Web sites or network links as it goes about its business ... Internally, the crawler must deal with huge volumes of data. Unless it has unlimited resources and time, it must carefully decide what URLs to scan and in what order. The crawler must also decide how frequently to revisit pages it has already seen, in order to keep its client informed of changes on the Web. [14]

One approach prioritizes the unvisited pages in the queue based upon the expected importance of the page. The assumption is that there will never be enough resources to crawl all of the Web, and certain pages are more important either because of their popularity or their ability to lead to discovery of additional important pages. Research suggests that such an approach increases the rate of crawl when compared to random- or breadth- ordered searches. [14]

Related work attempts to focus on a spider providing more than just the ability to crawl the Web. One such model would allow a spider to categorize documents automatically. The ARC system, for example, categorizes pages based upon their content as well as their perceived importance. [12]

## 2.3 Searching Through a Collection for Related Documents

For a user to issue a query, the first step is to translate a question from prose into a form that can be understood by a computer. This is often done with the assistance of either a person skilled at this art (e.g., a librarian) or by software that attempts to distill a natural language query into a Boolean form. Alternatively, the user can directly enter a number of key phrases or words with Boolean connectors like AND, OR, and NOT to describe the information that is desired.

Of course, a less skilled user might not choose the most relevant query terms. This is the so-called “vocabulary problem,” as a user might not have the vocabulary to describe the

desired information accurately. The classic solution is to store lists of synonyms for query terms as appropriate. Systems that update dynamically by defining new synonyms based upon past user success provide an additional improvement. [17]

## 2.4 Ranking Documents According to Relevance

The assumption is that a database will return a set of related documents based upon a set of Boolean connected keywords. An interesting challenge is to determine which of these documents are more important than others.

One classic approach is to rank documents based upon the frequency of the search terms appearing in the set of documents; that is, a document that contains the search terms twenty times would be considered more relevant than a document with only one occurrence of such a term. The advantage of this approach is that it is easy to implement and that minimal computation is required to determine importance.

However, such an approach will not produce unbiased results. It is quite easy to influence such a computation. Adding text (and in the case of HTML, making such text invisible to the casual user) that includes popular search terms many times in an effort to draw attention and advertise is a popular technique on the Web.

Another approach to ranking involves humans viewing the content of documents and individually determining their importance. However, the amount of information on the Web is typically too great to rank manually, and such a process leads to biasing. Thus, the majority of the current ranking schemes involve automated, algorithmic systems. These infer the importance of documents based upon the structure and organization of the Internet.

A number of different projects have investigated the structure of the Internet. Pirolli, et al. were some of the first to investigate the structural relationship between similar documents. Their work suggested that one could infer association by the links between documents on the Web. Such analysis could be more complex than simply observing direct hyperlinks. A more direct method would involve one reviewing the number of incoming and outgoing links for documents as a means for relating and categorizing documents. [23]

### 2.4.1 The HITS and PageRank algorithms

Two important methods of structure-inferred ranking were developed in the late 1990s. One hallmark of structure-inferred ranking was the Hyperlink-Induced Topic Search (HITS) algorithm, as developed by Jon Kleinberg. [20] The other is PageRank, developed by Brin and Page, and the current technology basis for Google. [9] Both attempt to discern which documents—i.e., pages on the Web—are the most relevant by using the surrounding hyperlink structure of the Web.

Kleinberg’s assertion is that relevance can be discerned by determining which documents carry the most *authority*. The hypothesis is that “[h]yperlinks encode a considerable amount of latent human judgment” and “this type of judgment is precisely what is needed to formulate a notion of authority.” [20] Both are based upon the assumption that when a document cites another (where on the Internet, a citation is a hyperlink between pages), there is a good reason for the citation. This latent reason provides a clue to which documents are important. One might look at all hyperlinks and observe which pages are cited most often, with the understanding that a hyperlink citation is a vote. Essentially, those documents cited most frequently are considered to be ‘better’ authorities than those documents that are cited less often.

PageRank proposes a ranking scheme based upon weight-propagation and eigenvectors, where a user randomly surfs the Web. The user selects an outgoing link for a given page uniformly at random or, with a probability  $p$ , decides to jump to a new page selected uniformly at random from the entire Web. The PageRank is the amount of time the user spends on a specific page, given such a random process. The amount of time ‘spent’ on a given page will go up if there are more ways to arrive at a given page. Thus, this method measures the number of citations for a given page.

HITS defines two scores for documents: an ‘authority’ score and a ‘hub’ score. Documents can serve as authorities, and contain valuable content, or be hub documents and link to valuable content. In addition to searching for documents of high authority, HITS looks for documents that serve as the best hubs. While more frequently cited documents often receive a higher authority score, the documents that *cite* the authoritative documents receive higher hub scores. In turn:

1. documents that are cited by documents with higher hub scores receive higher authority scores
2. documents that cite those with higher authority scores are in turn considered to have higher hub scores

Although this is a bit cyclic in definition, it can be mathematically expressed in practice: HITS’s goal is to assign a hub value and an authority value for each page. Higher values denote that pages are better hubs and are more authoritative, respectively. HITS does this by the algorithm in figure 3. A collection of hyperlinked pages is viewed as a directed graph  $G = (V, E)$ . The set of pages correspond to the vertices and the hyperlinks serve as the directed edges. A guess at the initial set of the hub and authority score for each page is made and successively refined based upon values of the surrounding pages. In other words, after assigning initial hub and authority scores to a collection, repeat the following iterative process: increase individual authority values based upon the hub values of adjoining pages, increase the hub values based upon authority values of adjoining pages, and normalize the hub and authority scores. Kleinberg has shown that as the number of iterations ( $k$ ) increases, the hub and authority value vectors converge.

But before running the Iterate computation, an appropriate set of pages must be provided. Kleinberg found that it best to have a root set  $S_\sigma$  that satisfied three properties: [20]

- i.  $S_\sigma$  is relatively small.
- ii.  $S_\sigma$  is rich in relevant pages.
- iii.  $S_\sigma$  contains most (or many) of the strongest authorities.

In practical terms, the set of documents that contains the query keywords will usually satisfy these properties and be a suitable  $S_\sigma$ . But such a set is not a good input to the HITS algorithm. The algorithm increases hub and authority scores according to the scores of the adjacent vertices. It is best to expand the initial subgraph to include adjacent vertices (and joining edges) so there can be a context for the algorithm. Kleinberg’s algorithm for expanding the initial graph to create a subgraph suitable for a input into HITS is described in figure 4.

An example collection is provided in figure 5. The gray documents (pages) and gray directed edges (links) are not considered in the Subgraph algorithm, but are provided for

Figure 3: Iterate Algorithm [20]

```

Iterate( $G, k$ )
   $G$ : a collection of linked pages (Web graph)
   $k$ : a natural number
  Let  $z$  denote the vector  $(1, 1, 1, \dots, 1) \in \mathbb{R}^n$ 
  Let  $H$  and  $A$  refer to hub and authority values
  Set  $H_0 := z$ 
  Set  $A_0 := z$ 
  For  $i = 1, 2, \dots, k$  {
    Let  $A_i^{<q>} = \sum_{(p,q) \in E} H_{i-1}^{<p>}$ 
    Let  $H_i^{<p>} = \sum_{(p,q) \in E} A_{i-1}^{<q>}$ 
     $A_i = \frac{A_i'}{|A_i'|}$ 
     $H_i = \frac{H_i'}{|H_i'|}$ 
  }

```

context. The bright yellow documents form the initial set  $R_\sigma$ .  $R_\sigma$  points to  $\Gamma^+(R_\sigma)$  via the directed blue edges.  $\Gamma^-(R_\sigma)$  points to  $R_\sigma$  via the directed green edges. When complete, the Subgraph algorithm returns the set composed of all pages in  $S_\sigma$ —i.e., all yellow documents. It also returns the associated edges—i.e., red, blue, and green arrows. Figure 6 depicts the collection returned by Subgraph on the example on figure 5.

Figure 4: Subgraph Algorithm [20]

```

Subgraph( $\sigma, \epsilon, t, d$ )
   $\sigma$ : a query string
   $\epsilon$ : a text-based search engine
   $t, d$ : natural numbers
  Let  $R_\sigma$  denote the top  $t$  results of  $\epsilon$  on  $\sigma$ 
  Set  $S_\sigma = R_\sigma$ 
  For each page  $p \in R_\sigma$  {
    Let  $\Gamma^+(p)$  denote the set of all pages that  $p$  points to
    Let  $\Gamma^-(p)$  denote the set of all pages pointing to  $p$ 
    Add all pages in  $\Gamma^+(p)$  to  $S_\sigma$ 
    If  $|\Gamma^-(p)| \leq d$  then {
      Add all pages in  $\Gamma^-(p)$  to  $S_\sigma$ 
    } Else {
      Add an arbitrary set of  $d$  pages from  $\Gamma^-(p)$  to  $S_\sigma$ 
    }
  }
  Return  $S_\sigma$ .

```

The goal of HITS is to use link information to infer authority. Some links, however, should not confer authority. For example, one should not infer anything from those links that are only designed to provide simple navigation. Kleinberg uses a few heuristics to minimize such impact. One is for a given page  $p$  to ignore all but one link from a single domain pointing to  $p$ . This avoids ‘collusion’ among the referring pages—e.g., the phrase

Figure 5: Collection of Documents To Be Used in HITS

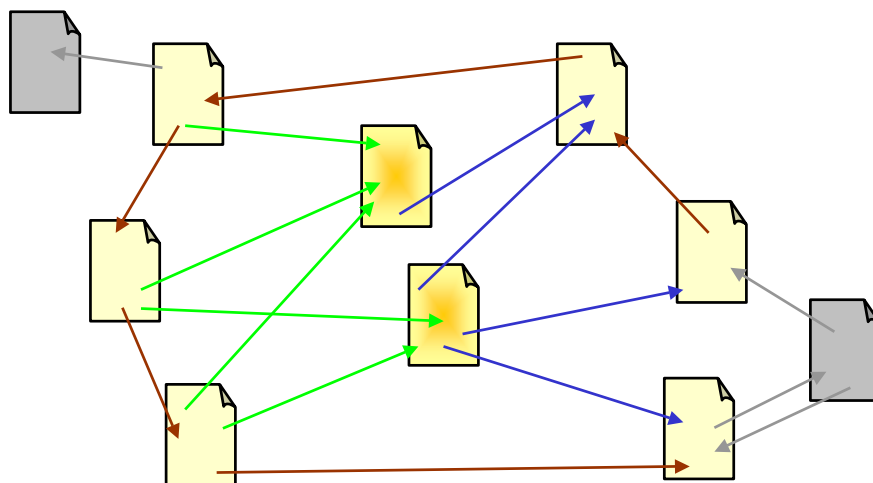
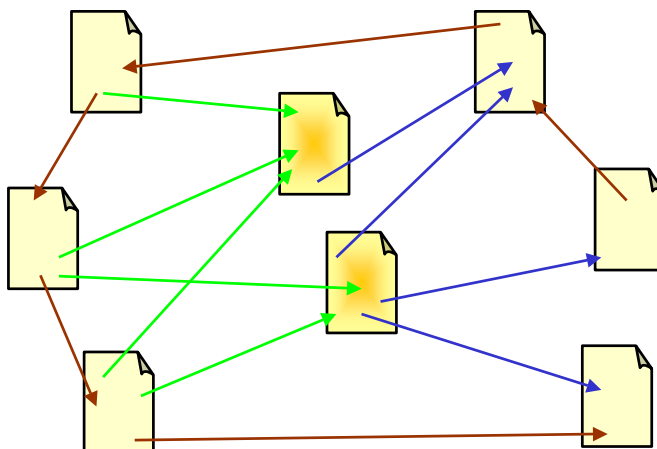


Figure 6: Collection of Documents for HITS After Running Subgraph



“‘This site designed by ... ’ and a corresponding link at the bottom of each page in a given domain.” [20]

Another heuristic is to minimize inclusion of links that are *intrinsic*—i.e., internal to a domain. Instead, only those links that are *transverse*—i.e., those links that cross different domains—are considered.

## 2.4.2 Additional developments in structure-based ranking

Recent research has attempted to resolve certain deficiencies in connectivity-analysis algorithms, particularly in HITS. One problem is characterized as *topic drift*. Although one might expect to find information specific to concise queries on a collection as large as the Web, there is usually a lack of plentiful hyperlinked documents to confer authority. As such, the Web does not provide HITS with an input sufficient to finding authoritative documents on very specific queries. The result is that we find authoritative documents on a generalized version of the original query: given a specific query like “California skiing”, HITS may produce authorities on more general results—perhaps general California tourism information.

The Clever system, based upon HITS, attempts to mitigate the problem of topic drift. The system assigns a weight to each hyperlink describing its importance based upon whether each hyperlink was intended only for navigation or if it can help determine whether the source page is authoritative and whether using the link will cause a drift from the original query. Specifically, these weights are determined based upon how similar the query terms are to the anchor text that surrounds links. So, for example, a link that points to *www.princeton.edu*, may have surrounding text as follows: “Princeton Univeristy, one of the colleges in the Ivy League, is located in central New Jersey.” We could reasonably infer that the destination is related to higher education, the Ivy League, and New Jersey. Thus, if we are performing a query for “Boston colleges”, the weight assigned to this link might be lower than one linking to various educational institutions in Cambridge, Mass. because this description is not directly related to the query terms. [13]

Bharat and Henzinger try a similar connectivity analysis algorithm improvement. [8] Their approach involves a score for each document considered in HITS based upon the similarity between the original query and the textual content of the document. There is an attempt to solve the implicit vocabulary problem: a query composed of a few keywords may not necessarily describe the desired documents effectively. Thus, the concatenation of the first 1000 words of the documents in the initial set of relevant documents is considered to be the expanded query,  $Q$ .

Instead of considering all nodes adjacent to the original set, a relevance score is computed for the adjacent nodes. The score for each adjacent document  $D$  is the mathematical distance between the vector composed of the frequency of terms in  $Q$  and the vector composed of the frequency of terms in  $D$ . (See section 3.3 for more on frequency vectors.) Adjacent pages are considered if the *similarity*( $Q, D$ ) is above a specific threshold. Thus, the expanded root set is a pruned version of the collection that would normally be the input to HITS.

Using similarity techniques, Bharat and Henzinger used HITS to solve two other problems. One goal was to minimize the influence of “automatically generated links” for navigation purposes; links with descriptions that appeared to be navigational were considered to be less important than links oriented more towards other content. The other goal was to minimize the influence of links that are “mutually reinforcing between hosts”—e.g., where

document  $A$  is linked to  $B$  and  $B$  is linked back to  $A$ . [8]

## 2.5 Performance and Architecture

One important aspect of a scalable search engine system is the way data is stored: if data is compressed and indirection is used in data structures, more computation resources are required for decent practical performance.

Consider, however, that Google currently indexes over 700 million pages on the Web; such a collection probably requires tens of terabytes of data just to hold the content. [10] However, this estimate does not provide for any indexing structures or link information for the calculation of the PageRank document importance algorithm that Google uses. Google's architecture contains components that store and access the collection in different manners.

When indexing the collection, the primary goal is to quickly determine which documents contain the query terms. As such, a database containing an index of keywords with pointers to the locations in original documents, along with metadata about characteristics of the original text (font size, HTML structure like `<H1>` tags, etc.) can be constructed. This allows one to issue a query with phrases that explicitly preserve the ordering of search terms. A separate module contains a database of all the hyperlink information in the collection. The PageRank computation requires this information for the connectivity analysis.

A theme in designing a search engine is ensuring fast retrieval. This involves complex data structures, and the requirement that most information be stored in both direct and inverted forms; the latter is required for connectivity computations and as a means to reconstruct the original data from the distributed nature of a given storage mechanism. To allow for storage of duplicate data, most of the data structures include hand-optimized bit arrangements such that billions of documents can be stored. For example, special two-byte structures store information about keyword location (general, anchor or title), capitalization, font size, and approximate position in document. The granularity of the information stored is not very fine, but storing only two bytes per term allows for the data structures in a "lexicon" of 14 million terms to remain in 256 MB of main memory. [10]

## 2.6 Search Engine Quality

Measuring the *quality* of a search engine is difficult, and inherently subjective. Some research has attempted to determine the quality of the index—i.e., does the index thoroughly represent the entire collection? Can an individual easily search the index for the documents of high authority? One approach suggests using a metric like PageRank to determine the quality of the documents in the collection and using this as a basis for judgement. [19]

Another metric for quality is the final result—i.e., can one find the desired information quickly and efficiently? This is the metric used in the Text Retrieval Conference (TREC) framework. The TREC evaluation begins with a detailed paragraph describing the desired information. After performing a search for this information, it is up to a blind panel to determine the quality of the results. The panel determines in a binary fashion whether each result is "relevant" (useful) and/or "correct" (what the query was looking for). [18]

### 3 Information Access, Storage and Retrieval Or, What does “similar” mean?

As a result of the research’s dependence on information—access, storage, and retrieval—two concepts must be understood. First, one must understand how documents are indexed. Second, as the main thrust of the research involves adding similarity information to the HITS algorithm, some attention must be given to what “similarity” is, and how this work uses existing knowledge in the field of information access.

#### 3.1 Indexing

An understanding of how documents are indexed is critical to understanding how search engines work. The indexing method will have performance implications, resource requirements, and allow for different types of searches.

Storing a complete cache of all documents would result in an index the size of the sum of all documents; this is usually not feasible with reasonable resource constraints. Performance might also suffer given the resources required to efficiently traverse such a large amount of content. However, were this practical, it would enable very complex queries. Fortunately, simple Boolean queries requiring the existence of query terms within documents have no need for a full content cache. Boolean searches for phrases, sentences, or query terms “near” one another would benefit from, however not necessitate, the context that a full document provides.

With these goals in mind, let us consider a simple method that only stores a list of all the unique keywords within each document. In many implementations this is the minimal amount of information that must be stored about a document to provide for a complete index. As such, it is assumed that this model falls well within any given resource constraints.<sup>4</sup> Effective use of multimap data structures (e.g., red-black trees or symbol tables) will allow for excellent performance. Unfortunately, this scheme only allows for simple Boolean searches. One would not be able to distinguish between a document referring to the product “Microsoft Windows” and one referring to the “large windows in building eight at Microsoft” since each logically contains ‘Microsoft’ AND ‘Windows’.

To create a scheme that allows for keyword context, an index must preserve the ordering of the original words in some form. Although storing complete documents is infeasible, creative use of list and map data structures allows us to achieve similar functionality with fewer resources. Using a map as a dictionary for words and keeping lists for each document describing the dictionary entries allows us to store context and locality—*ordered proximity*—at relatively low cost. [21, p. 94]

#### 3.2 Similarity

“Similar” is defined as: “1. Of the same substance or structure throughout; homogeneous; esp. 2. Having a marked resemblance or likeness; of a like nature or kind.” [5, ‘similar’] Thus, similar documents have the same substance or structure. Unfortunately, the definition of the substance and structure of a document is imperfect, although most definitions are “lexically based.” [21, p. 125] From this standpoint, documents that have common phrases are substantively the same.

---

<sup>4</sup>Making the assumption, of course, that any significant index will be allocated reasonable resources. “If you want to enter this ride, you must meet the minimum height requirements.”

Our research integrates content-based analysis into the structure-based analysis of the HITS algorithm. Instead of allowing HITS to consider all hyperlinks between documents, only a subset of the hyperlinks—one based upon the similarity of the source and destination documents—is considered.

### 3.3 Term Vectors<sup>5</sup>

Most lexicographical metrics measure the number of terms two sets have in common. At the most basic level a term can be considered to be a single word. But there is value in logically considering phrases—i.e., ordered tuples like pairs of words, triples, sentences, paragraphs—as terms. If we consider these more complex phrases as terms then we can better compare documents to determine lexicographic structure.

The natural response then is to consider a vector of these terms for each document. These are called *term vectors*; for the moment, assume that these term vectors contain the frequency of occurrence for each term. Long-standing standard mathematical approaches can be used to compare two of these vectors.

One approach is to use an angular measure like the *cosine measurement*. This measurement calculates the cosine of the angle between two vectors representing the documents. It is the inner product of the vectors, normalized by their lengths:

$$\sigma(D, E) = \frac{\sum_k (d_k \times e_k)}{\sqrt{\sum_k (d_k)^2} \times \sqrt{\sum_k (e_k)^2}}$$

$D$  and  $E$  are vectors representing documents.  $d_k$  and  $e_k$  are the value of term  $k$  in the respective documents.

Quantitatively, this “extrinsic” measurement focuses on the relative directions of these vectors to a fixed origin, and not on distance. [21, p. 85] A qualitative consequence is that a difference in term frequency plays less of a role than a difference in term existence. For example, a “one-paragraph announcement and an extensive, detailed paper about the same topic might be judged” similar. [21, p. 85]

Another approach is to measure the mathematical distance between the vectors. Quantitatively, this “intrinsic” measurement measures the absolute Euclidean distance in  $n$ -space between points. [21, p. 85] Qualitatively this measurement considers distance and does not focus on direction; in the worst case two unrelated vectors which are equally brief could be “similar.”

Generally, however, it turns out that the results are of the same “quality” regardless of the metric used for comparing term vectors! There will certainly be specific exceptions that are not accurately measured by one metric or another, but each metric is technically correct.<sup>6</sup> [21, pp. 85-86]

---

<sup>5</sup>This presentation of similarity term vectors closely follows the treatment provided by Korfhage. [21, pp. 125-130]

<sup>6</sup>This is a theme throughout the field of Information Retrieval. There tend to be many varied ways to quantify and analyze information; most are technically “correct,” all are “different,” but most are equally “good.”

### 3.3.1 Refining of term vectors

When using term vectors for comparison, the vectors should accurately represent the original files. The preference, however, is for comparisons to focus on the core differences between the documents and not on things that would reasonably be ignored. One might reasonably expect to ignore words like “*the, of, and, to, a, and in.*” In fact, these words tend to represent 20% of the words in most documents. Moreover, “the most common 250 to 300 words in English may account for 50% or more of any given text.” [21, p. 134] The common solution is to create a dictionary of such *stop words* and not include their occurrence in a vector representing a document.

A second improvement attempts to mitigate the problem of a vector representing an uncontrolled vocabulary. Namely, if one document contains “computers, computing, and computation” while another contains “computer, compute, and computations”, a pure vector approach would find these documents to be dissimilar. The classic solution is to use a *stemming algorithm*. An example of a stemming algorithm would be one that truncates the final ‘s’ in an effort to minimize the plural form of words.

The stemming algorithm that is typically used is one by Porter, which turns complex words into their stems more successfully than the simple algorithm to remove pluralization mentioned above. [24] The algorithm examines words looking for certain conditions, and replaces sequences of characters with alternate sequences. As alluded to above, one such condition might be ‘words ending with s’ with a rule to ‘remove the s’. There are dozens of rules, each relating to specific linguistic features of the English language. Here is an example of how this algorithm would stem the word ‘computationally’: *computationally* → *computational* → *computation* → *computa* → *comput*. [21, p. 136] The reader should notice, however, that other similar words will result in the same stem.

### 3.3.2 Comparing term vectors

Let us examine how to compare two vectors, where each vector contains a binary measurement of whether a document contains a term. Let  $D_1$  and  $D_2$  be these vectors, respectively composed of  $t_{1i}$  and  $t_{2i}$  (for  $i = 1 \dots N$ ). The notation in figure 7 allows the description, in aggregate, of how many terms two documents have in common. Figure 8 explains the notation graphically. [21, p. 126]

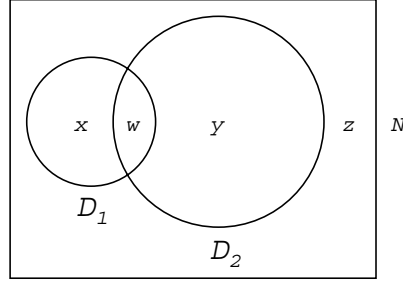
Figure 7: Notation for Term Vectors

$w$  = the number of terms for which  $t_{1i} = t_{2i} = 1$   
 $x$  = the number of terms for which  $t_{1i} = 1$  and  $t_{2i} = 0$   
 $y$  = the number of terms for which  $t_{1i} = 0$  and  $t_{2i} = 1$   
 $z$  = the number of terms for which  $t_{1i} = t_{2i} = 0$   
 $n_1 = w + x$   
 $n_2 = w + y$   
 $N = w + x + y + z$

With this notation, the following definition serves as a measurement of the similarity between two term vectors:

$$\delta(D_1, D_2) = w - \frac{n_1 n_2}{N}$$

Figure 8: Vector Notation Venn Diagram [21, p. 127]



This metric provides a way to determine the intersection between two vectors as well as a perspective based on the size of the vectors as compared to the universe of possible vectors. A value can take on positive and negative values.

The literature adds an additional factor to the unit of measurement described above. The unit is normalized in an attempt to measure the “distance” between the two documents. Not surprisingly, there are many possible different  $\alpha$  normalizations to use in the refined measurement  $\frac{\delta(D_1, D_2)}{\alpha}$ . There are at least a dozen coefficients that can be used, based upon simple computations like the vector angle between the two documents, to complex probability computations. [21, p. 128]

## 4 Experiment Design

### Or, What is HITS-SW?

The goal of this project is to improve the quality of relevant documents in content-query search engines. We believe the results of this work should be applicable regardless of the type of data set—digital library, particular domain on the Web, or the entire Internet—because all have hyperlink structures that can be used in HITS-SW.

The central focus of our research is evaluating the HITS-SW algorithm. HITS-SW is our modification to HITS that incorporates similarity weighting. The hope is that using similarity in a precomputed global manner independent of individual queries will address the problem of topic drift.

HITS and HITS-SW differ in the collection that serves as the input to the connectivity algorithm. HITS-SW assumes that each link on the Web has an associated similarity weight, measured by the similarity of the general content on the page that is the source of the link to the similarity of the link’s destination.

The input starts off with the same root set as normally occurs with HITS, but instead of expanding the root set to include the adjacent pages, a modification is added: including adjacent pages only if they are adjoined by a link that is within the range of acceptable similarity weights. Likewise, instead of including all edges between the pages in the set, only those edges within the acceptable range are included. This collection becomes the input to HITS-SW. The HITS-SW variant of the HITS Subgraph algorithm is described in figure 9.

An example collection is provided in figure 10. The gray documents (pages) and gray directed edges (links) are not considered in the Subgraph-SW algorithm, but are provided for context. The bright yellow documents are the initial set  $R$ .  $R$  points to pages via links  $k$ , denoted by directed blue edges. Pages point to  $R$  via links  $l$  and are denoted via directed green edges.

When complete, the Subgraph-SW algorithm returns the set composed of all pages in  $F$ —i.e., all yellow documents. It also returns the associated edges  $E$ —i.e., red, blue, and green arrows. Of interest now is the assumption that links within the specified range are depicted as *dotted arrows*. Thus, after running the HITS-SW Subgraph algorithm, the collection of yellow pages  $F$  and red, blue and green links  $E$  returned is the one depicted in figure 11.

The standard iterative HITS algorithm is run on this pruned collection and those items that contain the original query page are ranked—i.e., those pages in  $R$ . (Another possible approach would rank all pages that were a part of the expanded set  $F$ .)

Figure 9: Subgraph-SW Algorithm

Subgraph-SW:

- Add ids for all documents that match the query to sets  $R$  and  $F$
- For each link  $k$  with a source in  $R$  and with a weight in the specified range
  - Add destination of  $k$  to set  $F$
- For each link  $l$  with a destination in  $R$  and with a weight in the specified range
  - Add source of  $l$  to set  $F$
- For each document  $d$  in set  $F$ 
  - Add all links with  $d$  as a source and a destination in  $F$  to the set  $E$
- Return edges in  $E$  and nodes in  $F$

Figure 10: Collection of Documents To Be Used in HITS-SW

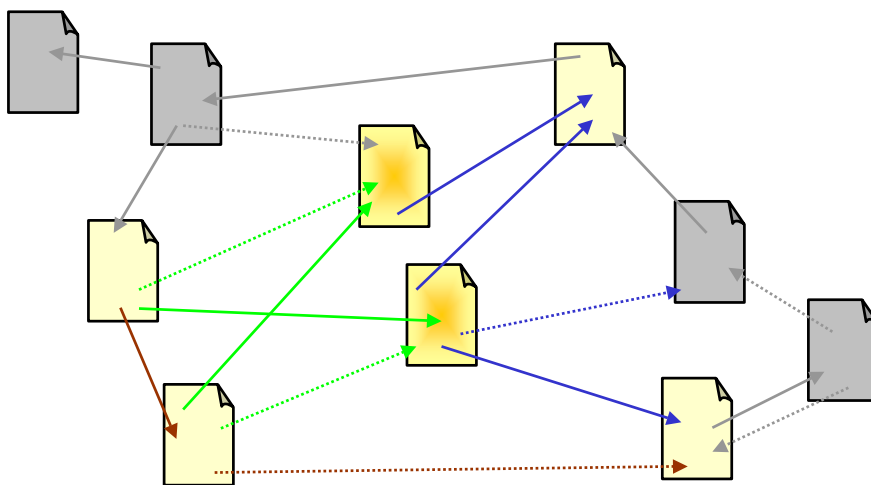
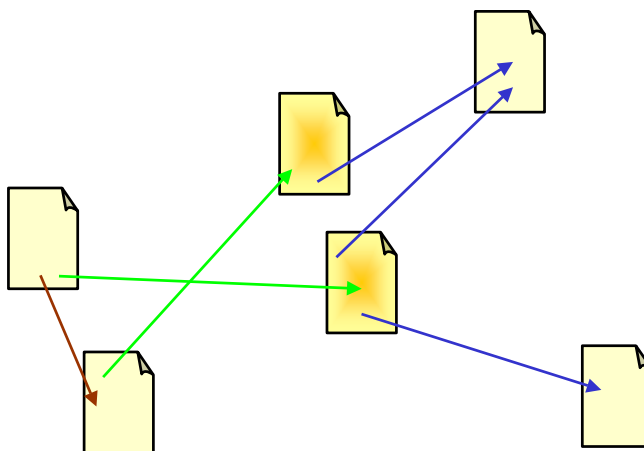


Figure 11: Collection of Documents for HITS-SW After Running Subgraph-SW



## 4.1 Policies

A corpus of data is required to test hypotheses about the use of similarity weighting in conjunction with the standard HITS algorithm. The first step is to gather hyperlinked documents for a corpus of content and associated link structure. Then, for each internal link, we must assign a weight denoting the similarity between the source and destination documents. With such a corpus precomputed, we can run tests to determine the efficacy of HITS-SW for a given query and a similarity preference.

### 4.1.1 Gathering Documents

Search engines for digital libraries usually have direct access to the documents within the library. As such, gathering documents is a relatively trivial task. Current Internet search engines, however, must crawl the Web looking for new pages; a time-consuming task. These spiders must also continually update their caches and indexes as the Web is not static.

Two data sets are used in this project. One is all the webpages residing on the server *www.cs.princeton.edu* (hereafter referred to as CS) and the other all the webpages within *princeton.edu*<sup>7</sup> (hereafter referred to as Princeton or PR). In many ways, the CS domain approximates a digital library such as a legal database; the pages within CS are generally narrow in topic, mostly relating to computer science. The hope is that the Princeton domain better approximates the Internet as a whole, as its content is more diverse.

Web search engines need to ensure that the cache of data indexed is updated periodically to ensure that decisions about relevant documents actually correspond to the real world data. In our research this is not a significant concern, and over time, the local cache of documents will undoubtedly fall out of sync with the real world. Such an assumption should not affect resulting data. Production-quality search engines are also reasonably tolerant of unreliable servers; attempts are made to contact servers that were unavailable when the pages were gathered to ensure the highest possible coverage. The implementation for our experiments did not reattempt to gather pages that generated errors. This simplification was justified by the low proportion of errors.

- 24,305 URLs were known in the CS domain, and 19,944 of these were successfully indexed.
- 145,854 URLs were known in the Princeton domain, and 138,300 of these were successfully indexed.
- 85,041 distinct singletons and 923,230 distinct pairs of keywords from the CS domain.
- 524,341 distinct singletons and 10,901,982 distinct pairs of keywords from the Princeton domain.
- 4,031,995 total keywords stored from the CS domain and 53,308,030 total keywords stored from the Princeton domain.
- 50,901 links from the CS domain had a destination outside of the desired scope.
- 399,386 links from the Princeton domain had a destination outside of the desired scope.

---

<sup>7</sup>Technically, this includes a little more than *\*.princeton.edu*, as it includes things like *dailyprincetonian.com* and *pppl.gov*. Of note, it does *not* include *\*.cs.princeton.edu*

### 4.1.2 Locating Relevant Documents

For the purpose of this work, a relevant document is one that contains ‘all’ of the query terms as defined by the user in a Boolean expression.<sup>8</sup> These queries consisted of only a few terms. This is a simple, generalized, and portable solution to this problem. It allows our research to focus primarily on ranking documents based upon relevance.

### 4.1.3 Ranking Documents Based Upon Relevance

Our goal is to integrate the advantages of the content-based ranking schemes with the structure-based algorithms. (See section 2.4.2 for more on the latter.)

It is worth distinguishing between the process of ranking the relevance of documents, and presenting the user with relevant documents. Part of the theoretical problem is determining which documents are more suited than others to the given query. There are, however, some complications: first, the user might receive too many results and suffer from information overload; second, those results that have the highest relevance might not provide an adequate amount of context.

Consider a search for a very detailed topic—e.g., “Gateway 2000 Solo Laptop 9100 Series Battery Life”. The result ranked highest may be a document with a concise answer (two hours), but without additional context, such a result may be inadequate. A user may question whether this result is for the standard or extended life battery. In some instances, a better document might be a more general one about the specific laptop in question or perhaps even Gateway’s comprehensive document about laptops.

On the Web, then, *root* pages corresponding to highly ranked documents might be more user-friendly, and are in fact *much* more desirable by many real users.<sup>9</sup> Most of these concerns relate to effectively communicating relevance information to the user. The goal of this research is not to focus so much on these issues of refinement, but rather to find a better mechanism for abstractly ranking documents.

## 4.2 Similarity

### 4.2.1 Binary Term Vector Approach

HITS-SW relies on determining the similarity between documents. Our approach uses the binary term vector style explained in section 3.3.2. Three types of terms are specifically stored: the individual words, pairs of words (after removing stopwords and applying a stepping algorithm), and URL references.

The assumption that pairs of words are sufficient to model longer phrases is made. This seemed suitable for testing, as we hypothesize that queries of phrases longer than pairs can be generalized from the results of pairs of words. Moreover, in our simple implementation for testing, the amount of storage required for pairs and phrases greater than three words would increase exponentially. (See section 5.1 for more information on our implementation.)

Given that we have three distinct types of terms: singletons (*1*), pairs (*2*) and URL references (*u*), we have seven different combinations (*1,12, ... ,12u*). However, we claim that only the following four are worth examining:

- *1* is the simplest approach and allows the examination of individual words.

---

<sup>8</sup>It should be noted that using a Boolean OR is usually the default in typical search engines, but we do not expect it to produce as interesting results.

<sup>9</sup>Google appears to cater to this desire. Many of their highly ranked results point to root documents.

- $12$  is slightly more complicated and provides data to determine whether inclusion of pairs of words is significantly different from just  $1$ . Although the expectation is that the number of pairs dwarfs the number of singletons, all terms are assumed to be equally relevant.
- $u$  allows similarity to be based solely on the number of URL references in common. The expectation is that this data will be substantially different from measuring similarity based upon content.
- $12u$  combines the terms from all three pools. Again, each term is given equal consideration despite the expectation that the number of words usually is much greater than the number of URL references.

Considering all terms equally important regardless of source will give varying results in certain boundary cases. For example, when documents have a great amount of content and few links, the number of  $1$  and  $12$  terms will be huge and the number of  $u$  terms will be small. Thus, the majority of terms in  $12u$  will be based upon  $1$  and  $12$  terms. The assumption is made that this disparity will not result in results that are significantly skewed.

#### 4.2.2 Simplified Similarity Metrics

Recall that there are over a dozen normalizing coefficients  $\alpha$  that are used with the basic unit of similarity. Many of these are similar, and can be categorized into three types of functions. The experiments use a representative sample from each category using the notation of section 3.3.2 and figure 7:

- Arithmetic Mean:

$$\alpha(E) = \frac{n_1 + n_2}{2}$$

- Linear Correlation:

$$\alpha(L) = \left[ n_1 \left( 1 - \frac{n_1}{N} \right) n_2 \left( 1 - \frac{n_2}{N} \right) \right]^{\frac{1}{2}}$$

- Yule Coefficient of Colligation:

$$\alpha(Y) = \frac{[(wz)^{\frac{1}{2}} + (xy)^{\frac{1}{2}}]^2}{N}$$

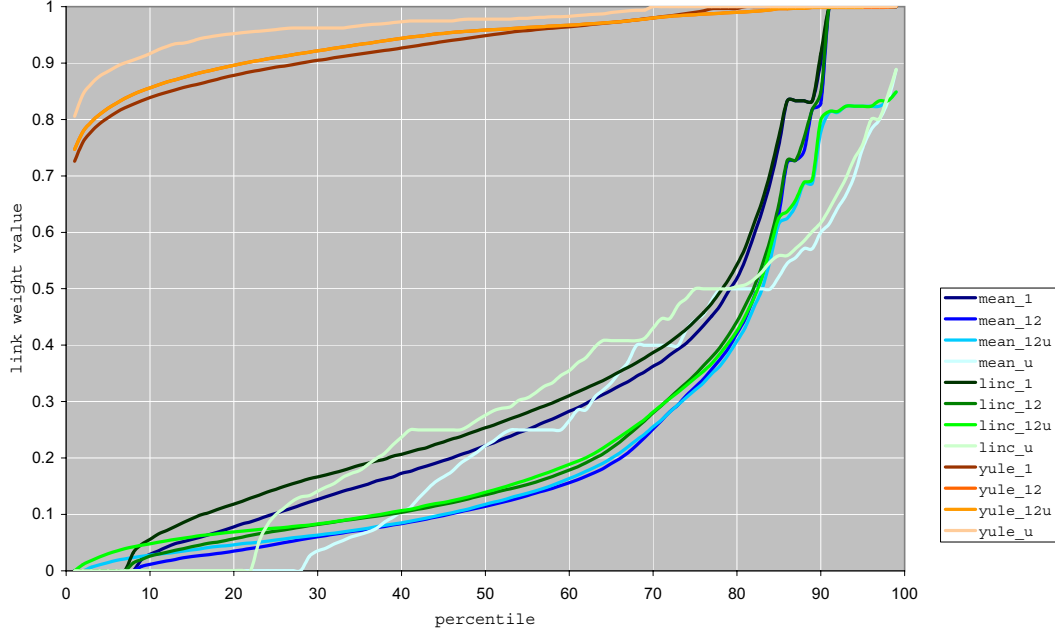
It can be generalized that  $\alpha(E) > \alpha(L) > \alpha(Y)$ , so the ordering of  $\frac{\delta}{\alpha}$  is the opposite. [21, p. 130]

#### 4.2.3 Candidates for Similarity Weights

With three normalizations and four types of terms considered, 12 distinct measurements of similarity, given any two hyperlinked documents, are defined. Given the quantity of data available for analysis, discarding six data points is reasonable.

Figures 12 and 13 describe the aggregate distribution of similarity weights assigned to hyperlinks. The  $y$ -axis corresponds to the assigned similarity weight, where a higher weight denotes increased similarity. The  $x$ -axis corresponds to the percentile in the cumulative distribution function. For example, in figure 12, about a third of the `mean_1` weights assigned are above 0.35.

Figure 12: Cumulative Distribution of Weights Assigned in CS Domain



Much of the data from the PR domain plateaus in the first quartile. Evidently many pages appeared to be very close to perfectly “similar,” as roughly a third of the links receive similarity weights close to one. Subsequently, only data obtained from the CS domain will be discussed, although the results from the Princeton domain are analogous.

One conclusion led us to discard all data that used the Linear Correlation coefficient,  $\alpha(L)$ . The reader should observe from figures 14, 15, 16, and 17 that in the aggregate, each of the three corresponding pairs of “linc” and “mean” ( $\alpha(L)$  and  $\alpha(M)$ ) tend to be similar. Namely, there is a constant offset between `linc_1` ( $\alpha(L)$  using singletons only) and `mean_1`, etc. This assertion was made on the aggregate data. The underlying assumption so far has been that the difference between each corresponding `linc` and `mean` data point would be a constant offset and that the aggregate graph was not the result of data canceling each other out. This assumption was reasonable and accurate. Figure 18 displays the difference between the corresponding `mean` and `linc` data points. As the values are negative, the values for `linc` are always larger (as expected). About 50% are within a small  $\epsilon$  of the original `mean` weight, and about 90% of all data points are within the original  $\|weight\|$ .

Because the differences between the `linc` and `mean` data were small, the expectation was that the differences in the final results based upon each of these data sets would be too small to warrant the effort of testing HITS-SW with both data sets. By using similar

Figure 13: Cumulative Distribution of Weights Assigned in Princeton Domain

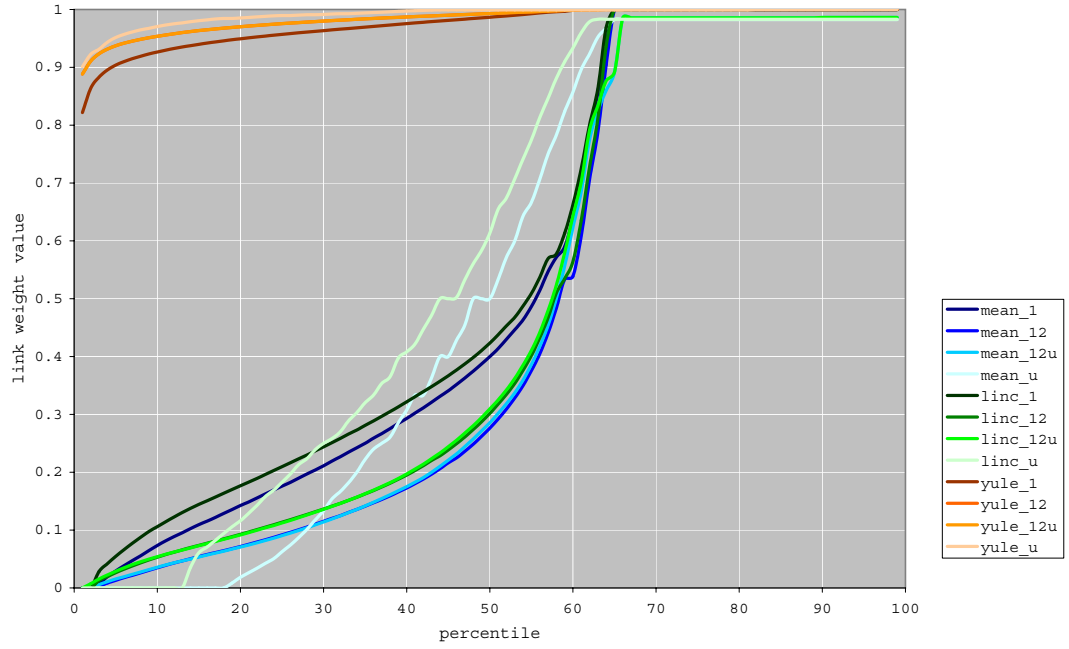


Figure 14: Cumulative Distribution of **mean\_1** and **linc\_1** Weights Assigned in CS Domain

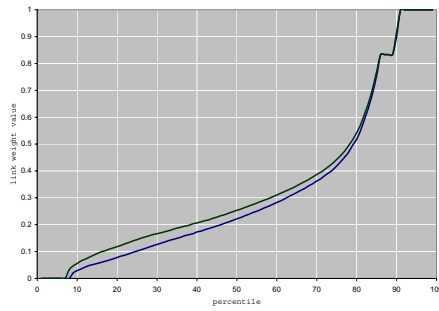


Figure 15: Cumulative Distribution of **mean\_12** and **linc\_12** Weights Assigned in CS Domain

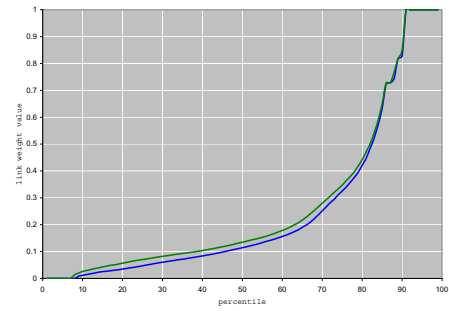


Figure 16: Cumulative Distribution of **mean\_12u** and **linc\_12u** Weights Assigned in CS Domain

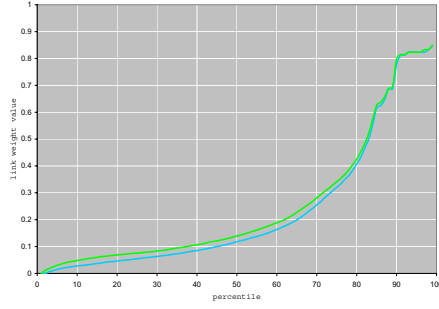


Figure 17: Cumulative Distribution of **mean\_u** and **linc\_u** Weights Assigned in CS Domain

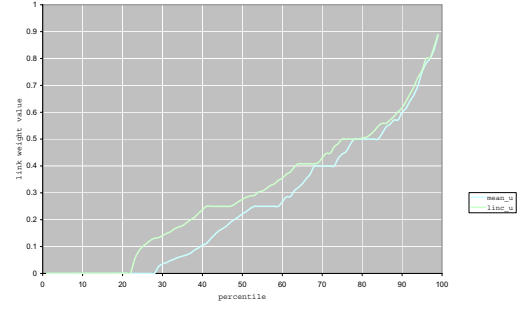
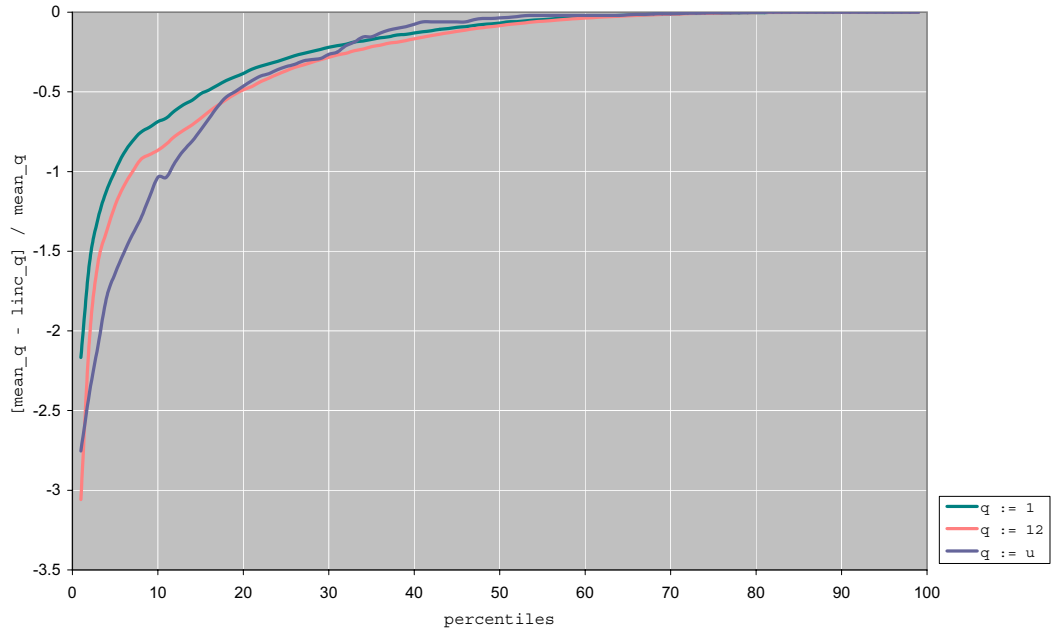


Figure 18: Differences Between Respective **mean** and **linc** Weights Assigned in CS Domain



graphs and logic, the “12u” data points were also eliminated for the `mean` and `yule`. This is illustrated in figures 19 and 20.<sup>10</sup>

Figure 19: Difference Between Respective `mean_12` and `mean_12u` Weights Assigned in CS Domain

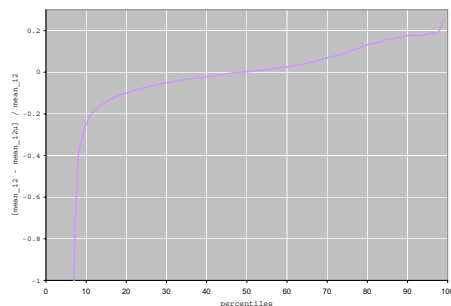
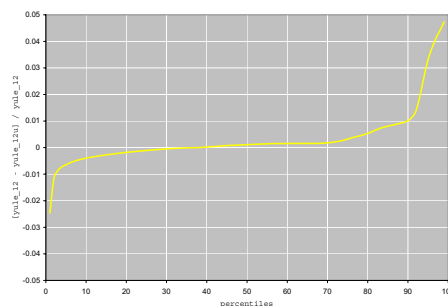


Figure 20: Difference Between Respective `yule_12` and `yule_12u` Weights Assigned in CS Domain



In the case of `mean`, the prediction whether `12` will be above or below `12u` cannot be made, and the great majority of the time the difference is very small—less than 20% of the original  $\|weight\|$ . There are, however, a few important outliers. About 7% of the weights showed a difference greater than an order of magnitude. Worse, about 3% differed greatly—by four orders of magnitude! For the `yule` data, the picture was a little clearer: the difference was always *extremely* small.

In general, the difference between `12` and `12u` was quite small. This is likely attributed to the significantly greater quantity of singletons and pairs than number of URL references. Initial inspection of this data led us to believe that we would be hard pressed to generalize differences in our final results based upon these minor differences because of the unpredictable nature of  $\|12\|$  and  $\|u\|$ : there were a small number of noticeable differences between `12` and `12u` in `mean`, and we could not predict whether or not  $12 > 12u$ .

#### 4.2.4 Final Similarity Weights

At the beginning of testing HITS-SW, six similarity metrics were at our disposal: `mean_1`, `mean_12`, `mean_u`, `yule_1`, `yule_12`, `yule_u`. Each query ran using each available similarity metric in HITS-SW, and in general the final ranked results were equally ‘good.’<sup>11</sup>

Appreciating the time constraints for this research, a decision to focus efforts on analyzing HITS-SW using the `mean` data was most logical. The differences in output between using `mean_1` and `mean_12` were generally unapparent. Thus, given that queries sometimes consisted of phrases, we decided to use the `mean_12` data in favor of `mean_1`. Although analysis using both the `mean_12` ( $m12$ ) and `mean_u` ( $mu$ ) data continued, most of the final results have focused on the  $m12$  data as the primary similarity weights in the use of HITS-SW.

<sup>10</sup>The reader should pay particular attention to the scale of figures 19 and 20, as they are quite different.

<sup>11</sup>This is certainly subjective. While we found that each produced a different ranked result, however the results seemed to have reasonably similar entropy values, and rankings that looked equally “interesting.”

### 4.3 Obtaining and Measuring HITS-SW Results

The data from two similarity metrics, `mean_12` and `mean_u`, were used in the HITS-SW queries. An additional parameter describing the desired similarity range is required to use HITS-SW to rank results. This range was specified as a real number between 0 and 1. Including the entire range  $[0, 1]$  effectively means running HITS-SW as pure HITS, as it is not based upon similarity weights.

#### 4.3.1 Testing HITS-SW

Our tests of HITS-SW used two ranges:  $[0, m]$  (“lo”) and  $[m, 100]$  (“hi”) where  $m$  is the value that denotes the edge that is the median weight. This use of two ranges tests two extremes of HITS-SW, and using the median made more sense than choosing 0.5 because this will account for the links having a disproportionate number of similar or dissimilar weights. (In fact, for the data used in these experiments, the median tended to be lower than 0.5, although it varied based upon the specific similarity metric used.) The median was based upon the aggregate data shown in section 4.2.3; it varied between data from the Princeton domain and the CS domain.

#### 4.3.2 Measuring Results

Our initial observations of the *hi* and *lo* queries suggested certain trends in the results’ ranking. We developed quantitative metrics called *entropy measurements* to compare HITS-SW ranking to that of HITS.

Our expectation is that users will concentrate mainly on the ranking of the top results and pay attention to the top 20 ranked items, with particular focus on the top 10. In other words, people consider the arrangement of the top 10 to be the most important but do inspect the next 10 results as a whole. Thus, the quantitative metric only considers the top 20 positions even though it is possible that HITS-SW produced wildly different results for ranking less important items. Although only the top 20 results are scrutinized, HITS-SW does assign a rank to all the results based upon the authority score.

As often seen in information retrieval, there is not one explicit way to measure the difference between the two permutations. While each metric is particularly good at capturing a certain “type of difference,” each usually misses equally important types. The following metrics measure the difference between two orderings  $P_0$  and  $P_1$ . The hope is that when used together they accurately describe the differences in orderings.

There is no obvious way to combine all of these metrics into a single scalar value. Each measurement provides a different way to compare two permutations and thus produces slightly different results. Moreover, these metrics do not exhibit desirable properties like obeying the triangle inequality theorem. As such, their use is limited to comparing HITS-SW rankings to the corresponding HITS ranking, always setting  $P_0$  to be the HITS ranking.

1. *elements orig* and *new* refers to the total of elements in  $P_0$  as well as the total elements in  $P_1$ . Comparing the differences shows how much HITS-SW decreased the number of ranked results.
2. *inversions t10* counts the number of inversions seen in the top 10 elements in  $P_1$ . An inversion is when a pair of elements in a permutation are out of their natural order. Inversions describe to what extent the new top 10 elements are ‘mixed up.’

Let  $i_1 i_2 \dots i_n$  be a permutation of the set  $\{1, 2, \dots, n\}$ . The pair  $(i_k, i_\ell)$  is called an inversion if  $k < \ell$  and  $i_k > i_\ell$ . [11, p. 87] For example, the sequence  $\{10, 5, 9, 11, 15, 3, 1, 2\}$  would have 19 inversions as there are 5 elements less than 10 that follow 10:  $\{5, 9, 3, 1, 2\}$ , 3 after 5:  $\{3, 1, 2\}$ , 3 after 9:  $\{3, 1, 2\}$ , 3 after 11:  $\{3, 1, 2\}$ , 3 after 15:  $\{3, 1, 2\}$  and 2 after 3:  $\{1, 2\}$ .

3. *inversions orig* counts the number of inversions seen by observing only those elements in the top 10 of  $P_1$  that were also in the top 10 of  $P_0$ . This should provide insight into whether the original top ten elements were reordered or if new elements were promoted to take their place.
4. *intersection t10* notes the number of elements in the top 10 of  $P_1$  that were also in the top 10 of  $P_0$ . This shows whether the elements in  $P_1$  are new or not.
5. *intersection t20* notes the number of elements in the top 20 of  $P_1$  that were also in the top 20 of  $P_0$ . It is important to determine the intersection of the top 20 because of the assumption that while the ranking of elements 11-20 do not matter, the fact that they are in the group 11-20 is important.
6. *distance* measures how far each element in the top 10 of  $P_1$  has traveled to arrive in the final destination. This metric helps determine whether the top 10 elements in  $P_1$  were formerly ranked as not very important, and thus highly promoted. It is computed by the following equation:

$$\frac{1}{2} \sum_{i=1}^{10} \log_2 |i - P_0^{-1}(P_1[i])|$$

The factor of  $\frac{1}{2}$  accounts for the symmetry involved in such a measurement. The logarithmic scale helps ensure that highly promoted items do not dwarf other promotions.

### 4.3.3 Creating a control group with random similarity weights

Our initial observations of HITS-SW produced results that were qualitatively and quantitatively different from HITS.

Comparing results tends to be a subjective task, and it is difficult to determine what precipitates changes. To determine if the results are a factor of intentionally chosen similarity weights, we attempted to reproduce HITS-SW-like results from arbitrary similarity weights. Recall that tests on HITS-SW focused on running queries and considering either those links that received a similarity weight above or below the median weight. Therefore the goal was a test that could simulate how the HITS-SW queries would consider up to half of the edges. The test would simulate the creation of an arbitrary weighting function, and randomly select 50% of all edges, and consider them as “within the threshold” for HITS-SW. By considering an arbitrary subset of the links as input to our algorithm, we could determine whether our results were because HITS-SW only used a subset of links, or whether (as we hope) HITS-SW focused on a well-chosen subset of links.

For each query, 25 random tests served as a baseline for comparison to HITS-SW queries. Figure 21 contains pseudocode describing the process used to run these trials. The same arbitrary weighting function was used across queries. Thus, the same edges would be *masked*

out—and thus not available for consideration—for a specific test across all queries. Thus, for example, the edges masked in test one would be for all queries. This masking determined the topology of the graph. As HITS (and therefore HITS-SW) is based purely on the topology, this masking effectively caused a change in how ranking was determined. As this consistent masking caused specific topologies to develop, the hope was that documents ranked in separate queries would receive consistent rankings because of the consistent topologies.

Figure 21: Pseudocode for Random Similarity Function

$R_{i,f}$ : function  $R_i$  that *masks* out  $(1 - f) \times$  edges  
 $E$ : all edges in the universe  
 $Q_j$ : A query  
 Let Subgraph-SW-R model Subgraph-SW (figure 9) with modification:  
     “weight in the specified range”  $\equiv$  “edges not masked”  
  
 For  $i = 1, 2, \dots, 25$  {  
     Apply  $R_{i,0.5}(E)$  to mask 50% of edges  
     For  $j = 1, 2, \dots, n$  {  
         Run Subgraph-SW-R and HITS-SW for query  $Q_j$   
     }  
 }

## 5 Implementation

### Or, How do we collect 8 GB of data?

One goal of our research was to create a set of data that can be used to test hypotheses about use of similarity weighting in conjunction with the standard HITS algorithm. The first step is to gather hyperlinked documents for a corpus of content and associated link structure. Then, for each internal link, we must assign a weight denoting the similarity between the source and destination documents.

With such information precomputed, tests determine the efficacy of HITS-SW for a given query and a similarity preference. Standard retrieval techniques are used to find relevant documents for the query. Then HITS-SW runs for the similarity preference on these relevant documents.

In this implementation, Perl[3] was used to write a majority of the code; the software ran on local Solaris servers. The data was stored in a relational Oracle database accessible over the network. Our components can be divided into those designed to gather and prepare data, those related to issuing queries, those related to running the HITS-SW algorithm, and those related to analyzing the output. The expectation is that the components designed to gather and to prepare the documents are run (in series) before any queries are issued.

#### 5.1 Database Schema

Central to the organization of the data is a table that stores a mapping between all of the URLs comprising the designated corpus and the assigned numeric ID. A number of flags for storing state are also stored in this mapping. The definition of this table can be found in figure 22.

Figure 22: SQL table definition: urls

```
CREATE TABLE  urls(  
    id          INTEGER      NOT NULL,  
    url         VARCHAR(256) NOT NULL,  
    universe    INTEGER      NOT NULL,  
  
    link_state  INTEGER,  
    content_state INTEGER,  
  
    PRIMARY KEY (id)  
)
```

Field `content_state` stores information regarding whether the content pointed to by this `url` has been gathered and whether an error was encountered while doing so. Field `link_state` stores information regarding whether the links with this `id` as a source have been assigned weights yet. Field `universe` allows for multiple corpuses, each containing its own set of URLs.

All other tables ignore the concept of *universe*. Separate table instances are used instead of a field within each table designating the corpus. A naming convention like `tablename_one` or `tablename_two` for universes 1 and 2 is used.

This approach is slightly more inconvenient for programmers because accessing SQL commands must be modified to reflect the varying names for tables. It does, however, allow for a few significant advantages. It primarily allows for faster access: by using disjoint tables, simple queries that do not involve existing indexes do not have to sort through data

in another corpus that should clearly be ignored. It also eases maintenance as it enables the gathering of documents for one set of data to run interrupted even if another is off-line for maintenance.

Unique words and phrases found when gathering content are stored in the table defined in figure 23. Links destined within our desired scope of documents are stored. Subsequently, they will have similarity weights assigned to them. The table that stores this data is defined in figure 24. Although they are unused in HITS-SW, links that point to documents that do not map to an `id`—i.e., those links that are outside our scope—are stored in the table defined in figure 25.

Figure 23: SQL table definition: content

```
CREATE TABLE content(
  id          INTEGER      NOT NULL,
  phrase      VARCHAR(80)  NOT NULL,
  type        INTEGER      NOT NULL,

  PRIMARY KEY (id, phrase)
)
```

Field `type` denotes whether a phrase is a single word or pair.

The given schema is optimized for bulk loading of data. With the exception of the implicit indexes created by the primary key constraints, there are no indexes that need to be maintained when documents are gathered as the database is bulk loaded with content and link data. In fact, although the database is optimized for loading data, it is not optimized for any database queries.

Once the database contains all the content and link information, two indexes are created. One index is on the `ids` within the content table and the other is on the `phrases` in the content table. These allow the efficient retrieval of the information necessary to determine similarity.

## 5.2 Seeding the Database

The database was first populated with all the URLs within the scope of consideration. In our case, the URLs were extracted from the local search engines corresponding to the domains chosen.<sup>12</sup> Extracting from these two sources ensured that all URLs were known to point to valid resources. The spiders in these production search engines respected the privacy of published documents by obeying the robots exclusion protocol. [2] As such, our data only consisted of documents intended for public access.

Before these URLs could be used as seeds, some processing was required to make the format more uniform. URLs are typically composed of a protocol (e.g., HTTP), a site (e.g., `www.princeton.edu`), a port (e.g., 80), a location on the site (e.g., `/gleeclub/`), a file (e.g., `index.shtml`), and possibly an anchor (e.g., `#section2`). The location and filename are typically case sensitive, whereas the other information is not.

For this study, it was desirous to index all textual content, but many of the URLs provided did not fit this criterion, as they pointed to binary files of varying forms. There

---

<sup>12</sup>These extracted URLs—from the Verity search engine used at Princeton and the Infoseek search engine used in the CS department—were stored as a formatted textfile.

Figure 24: SQL table definition: links

```
CREATE TABLE links(
  id1          INTEGER      NOT NULL,
  id2          INTEGER      NOT NULL,

  link_state   INTEGER,

  v_plain_1    REAL,
  v_plain_12   REAL,
  v_plain_u    REAL,
  v_plain_12u  REAL,

  v_mean_1     REAL,
  v_mean_12    REAL,
  v_mean_u     REAL,
  v_mean_12u   REAL,

  v_linc_1     REAL,
  v_linc_12    REAL,
  v_linc_u     REAL,
  v_linc_12u   REAL,

  v_yule_1     REAL,
  v_yule_12    REAL,
  v_yule_u     REAL,
  v_yule_12u   REAL,

  PRIMARY KEY  (id1,id2)
)
```

Fields with the suffix `_1` refer to the weight that only considers phrases of individual words when assigning a similarity; the suffix `_12` refers to the weight that considers phrases of individual words as well as pairs; `_u` refers to the weight that only considers the links; finally, fields with the suffix `_12u` refer to the weight that considers singletons, pairs, as well as links when determining similarity.

The keyword `plain` refers to the unnormalized weight as explained in section 3.3.2. The keyword `mean` refers to the weight as normalized by the “arithmetic mean” coefficient, `linc` refers to the “Linear Correlation” and `yule` refers to the “Yule Coefficient of Colligation”. [21, p. 129]

Figure 25: SQL table definition: otherlinks

```
CREATE TABLE otherlinks(
  url1         VARCHAR(256) NOT NULL,
  url2         VARCHAR(256) NOT NULL,

  link_state   INTEGER,

  PRIMARY KEY  (url1, url2)
)
```

was no easy way to remove these undesired URLs from the set, but the following heuristics were used to make the set of URLs we received more uniform:

1. Only keep URLs pointing to resources accessed by HTTP.
2. Remove any explicit mention of port 80.
3. Remove all anchors from URLs.
4. Change the protocol and site to lowercase, but leave the case sensitive location and filename alone.
5. Unescape characters.
6. Truncate typical ‘default’ filenames like *index.html*, *index.shtml*, *index.htm*, *default.html*, *default.htm*, *default.shtml*, *index.php*, *default.php*<sup>13</sup>
7. Append a forward slash to URLs that appeared to end in a directory (e.g., *www.princeton.edu/cit* → *www.princeton.edu/cit/*).
8. Ensure that these changes do not result in a duplicate entry.
9. Finally, attempt to remove all URLs that do not end in a known text extension.<sup>14</sup>

### 5.3 Core Library Functionality and Performance Designs

There is a core library written in Perl that is designed to provide many of the services required for infrastructure. There are settings for configuration of all the scripts and information necessary for customizing the software for each specific universe of data.<sup>15</sup> Accordingly, there are different ways of interacting with the database to access the separate tables for each domain of data.

There are also a number of text processing tools. There are functions for converting URLs to lowercase in the manner described above, as well as many functions to convert text into *keywords*. Namely, the following techniques are used:

1. Remove all non-alphabetical characters. The assumption was that people will not search for non-alphanumeric characters and that it would be reasonable to simplify the data and not include numbers for the scope of this study.
2. Convert most words to lowercase. If a word appears to be an acronym (at least the first two characters are uppercase), it was left as is.
3. Use Porter’s stemming algorithm to turn words into their root form.
4. Remove stopwords. A classic list of stopwords, with the addition of a few Princeton-specific words (e.g., “Princeton”), was used. [1]

---

<sup>13</sup>This truncation was easier on the CS domain as we could quickly determine which one or two were used on the single webserver. Within the Princeton domain, it was more difficult as many webserver were used throughout Princeton, and we could not guarantee that we had eliminated all possibilities.

<sup>14</sup>With the use of a regular expression, we looked at all the unique final five characters sequences and judged, by inspecting a sampling of the URLs whether such an extension referred to a binary file. This approach had mixed success.

<sup>15</sup>Our data is segregated into two universes—the Computer Science domain and the Princeton domain.

5. Extract all unique words and pairs of words.

Finally, there were additional functions to facilitate the partitioning of URLs in the database into  $n$  sets.<sup>16</sup>

## 5.4 Database Information

Here are some statistics about the data we stored in our database:

- 7.35 GB of data and indexes.
- 0.9 GB of the total database corresponds to the CS set of data. Of this, 0.7 GB is actual data while 0.2 GB corresponds to indices.
- The remaining 6.45 GB of the total database corresponds to the Princeton set of data. Of this, about 4.0 GB is actual data while 2.45 GB corresponds to indices.

## 5.5 Performance Concerns

In the design of each module, there was always the decision whether information should be locally cached to prevent additional database queries and network usage. The scope of caching varies from module to module.

Due to the high frequency with which each component would need to translate between the arbitrary `id` assigned to each URL and the actual URL, it made sense to cache such information locally. Although only about 10 MB were required on disk, storing such information in a hashtable within Perl required about 40 MB of memory. This was an interesting finding, given the usual expectation that a hash table requires about 150% space usage. Our hypothesis for this 400% expansion is that Perl does not efficiently store data given that data are inherently untyped. Running five processes per machine uses about 200 MB of memory. But in the larger scheme, where each machine had 8 GB of memory and 20 GB of swap, running five processes per machine seemed reasonable.

Another module required access to all of the similarity weights for all of the links. The first version of the software did not cache this locally, as it was on the order of 100 MB on disk. Such a version took ten days to run a process, which was unacceptable. Caching such information locally so that we would not have to issue many queries brought the running time down to about ten minutes.

## 5.6 Gathering Data from the Web

This *lynxwrap* module had the task of downloading and parsing all pages from the Internet, and storing the keywords in the database for subsequent similarity computations. The module was designed so that many instances could be run in parallel, and it took advantage of the core library's ability to partition the set of URLs into multiple portions. Each instance of the *lynxwrap* software received instructions upon startup regarding which particular partition it would pursue. This parallelization was important because a fair amount of time and computation was required to initiate a network connection and then to download, render, and parse the content.

---

<sup>16</sup>For the purposes of our implementation,  $n$  was set to 10. We would usually distribute these 10 tasks across two machines, as we were limited by the number of simultaneous network connections allowed per machine to the database.

The software used a modified version of lynx, the terminal emulation Web browser. [4] The modifications allowed it to take a URL as an argument and output both the hyperlinks found within the document and a rendered version of the HTML suitable for parsing.

One particular complication encountered was reliably downloading unreliable information. As pages are stored in a distributed manner, there is no guarantee that they will be available for download. Taking this into consideration, spiders should make several attempts to repeat downloading pages which exhibit errors (e.g., server is down). Part of the complication was determining all the possible errors that can occur with such varied input as the Web provides. It was usually easy to determine if the expected page no longer existed, as HTTP error codes existing for such problems are easily obtainable. Typical problems encountered related to correctly capturing the content and turning it into keywords. If a document contained zero information, attempts were made to exclude it from the database. After clean-up of a document's content (stemming, condensing, etc.), a secondary check ensured the document was still significant. Results from an HTTP GET were at times not reproducible: a page would often be returned as blank, but without an error. The cause of this problem is unknown, but a work-around was to repeat the HTTP GET if this error was encountered.

Another problem stemmed from the imperfect heuristic used to avoid indexing binary files: occasionally, as a result of a binary file, a module 'found' a disproportionately large number of keywords. The simple solution was to ignore any page that had a suspiciously large number of keywords.<sup>17</sup>

This module also had the task of loading the database with the hyperlinks found in the documents. If the hyperlinks found linked to a page that was one of the seeds (and thus had an id preassigned), they would be inserted into a separate bucket from those that pointed outside the domain we wished to download. Pseudocode for the lynxwrap module is described in figure 26.

Figure 26: lynxwrap pseudocode

```

For each URL, i, in our partition of the set of unvisited URLs {
  Download i, reporting HTTP errors
  Update database with the links in i, separating into appropriate tables
  Turn formatted HTML of i into minimal keyword text format
  Update database with singleton and pairs of keywords
}

```

## 5.7 Assigning Similarity Weights to Hyperlinks

This *statwrap* module computed the similarity weights for all the internal links that were found. This module was another process that could run in parallel with other instances of itself. The ability to run in parallel was important because there was usually eight times as many links as original URLs, and processing each URL required a number of database queries to retrieve the information necessary to perform the similarity computations.

---

<sup>17</sup>For our purposes, the threshold was set to be at least 8000 distinct individual words or at least 70,000 distinct pairs of keywords. These parameters were arbitrary based upon empirical results showing that they precluded including binary files but did not seem to affect the inclusion of any text files we wished to index.

The expectation was that this module would run once the lynxwrap model had downloaded all the content. This was required because part of the configuration input was information describing the total number of distinct keywords found in the domain.

To minimize the number of database queries, this software ran the similarity comparisons as two nested for loops. This was a reasonable improvement over no caching, and required little programming effort. Although better algorithms and caching structures existed, this module only needed to be run once, so the inefficiency was acceptable.

Each similarity metric built upon the same mathematical foundations. To compare two documents  $D_1$  and  $D_2$ , the relationship between the keywords in the set composed from  $D_1$  and those composed from  $D_2$ , had to be known. Such computations could be used in determining each similarity weight. Pseudocode for the statwrap module is described in figure 27.

Figure 27: statwrap pseudocode

```

For each URL,  $i$ , in our partition of the set of unweighted URLs {
  Retrieve all keywords in  $i$ , and store for computations
  For each URL,  $j$ , that is the destination of source  $i$  {
    Retrieve all keywords in  $j$ , and compute the similarity( $i, j$ )
    Update database with results, and mark  $\langle i \rightarrow j \rangle$  as finished
  }
}
```

## 5.8 Preparing a Query-Specific Collection for HITS-SW

This module would query the database and prepare a collection of related documents for input into the HITS-SW algorithm. While this module could be run in parallel with other instances, it did not need to partition any information in the database.

This software had two tasks: first, to find the relevant documents—i.e., those that contained the query terms (for most of our queries, multiple keywords were logically ANDed together); second, to expand the root set in accordance with the HITS-SW principles. The first of these tasks required access to the database; caching the information would not be efficient because the amount of data was large and diverse as compared to the number of times we would use it. The second task required access to the similarity weights assigned to all the links, as each must be inspected to determine whether the associated weight is within the specified range. This weight information can be locally cached.<sup>18</sup>

## 5.9 Hubs and Authorities Computation

This is a C implementation of the HITS iterative algorithm created by Jonathan Baccash. [7] This module alone is a pure HITS implementation, but combining it with the output of the Subgraph-SW module yields an implementation of the HITS-SW algorithm.

---

<sup>18</sup>In fact, caching such information resulted in saving at least two orders of magnitude of time!

## 5.10 Manipulating Ranked Output

Scripts manipulated the output to better communicate the results of the HITS-SW algorithm. In addition to reformatting the document in a manner that is easier to comprehend (e.g., adding URLs to the arbitrary `ids`), analysis of the output included a comparison of the reordered URLs to the analogous output from the standard HITS algorithm for the same query. Additional computations demonstrate quantitatively how HITS differed from HITS-SW. See section 4.3.2 for further explanation.

## 5.11 Random Weighting Function for Control Group Tests

A function that could select an arbitrary subset of hyperlinks for considering in HITS-SW was required. These elements were stored in an array, and the goal was to mask off half of the elements for use. The approach taken was to mask off the first half of the elements and then randomly shuffle the array, using the Fisher-Yates shuffle. [15, pp. 121-122]. The shuffle algorithm used a pseudorandom number generator, seeded with a distinct random value for each of the 25 trials conducted.

## 5.12 Security, Performance, and Political Concerns

Because of the massive size of the Web, spiders must be able to efficiently gather pages at an extraordinary pace. This presents a security problem: most public servers do not expect to have a client download all of their content simultaneously. (This is a real possibility with multiple crawlers running in parallel.) Thus, a spider must be a respectful Web citizen and not overwhelm servers. A simple solution would be to interleave requests for data between different servers. [10]

Components have many instances running at once, each using tens of megabytes of memory for hashes of data and hours of CPU time for downloading content, computing similarity weights, or running the HITS computation. Such resource usage did not always go unnoticed. It is common for University users to casually use days of CPU time and large amounts of memory for mathematical computations in a program like matlab. Running an `a.out` program for an introductory computer science course that is stuck in an infinite loop also usually goes unnoticed. However, running resource-intensive programs with other names is unusual. This is what much of our software was doing.

When running software that downloaded all content from all the servers in the Princeton domain, we wanted to ensure minimal impact on others. An admonition was given to the proper University departments that this software would be running and that the speed of the programs could be decreased if the impact was a burden. This announcement brought even more attention to the ten or so instances of `lynxwrap` running on the campus servers.

At about the same time, there were a number of security incidents involving illegal port scanning and connections originating within Princeton to external sites (e.g., national science laboratories). While the timing of such incidents was fortuitous, until the causes were determined several days later, negative attention was brought to the HITS-SW processes being run. The simplest approach, given no impending deadline for data collection, was to suspend the collection of data until innocence could be demonstrated.

## 6 HITS-SW Findings

### Or, As a result, did the sharks get smarter?

Although it is difficult to make a blanket statement like “one should always use HITS-SW with a certain similarity parameter,” the results of using HITS-SW are indeed interesting, and worthy of analysis. One difficulty is objectively observing patterns that are generalizable.

The quantitative metrics for the entropy provide insight into a reordering, but do not provide an understanding of the specifics of what happened in a given reordering. By observing the name and location of specific URLs, one can gain some insight into the content of the document. The original content pointed to by the URLs allows for the best understanding of what a reordering means.<sup>19</sup>

Though one hopes to correctly infer the rationale behind reordering the documents based upon an investigation of the similarity weights and the content of the surrounding documents, there is no guarantee of this procedure. The graphs of similarity weights are usually quite complex; trends in specific queries usually cannot be generalized to explain reordering in other queries. As such, many conclusions remain quite subjective.

#### 6.1 Randomized Control Groups Are Unlike HITS-SW Tests

The *hi* and *lo* tests using HITS-SW are qualitatively and quantitatively unlike the randomized tests run. (The reader may observe the data in appendix A.) The great majority of the random results do not look like the *hi* and *lo* tests. Aggregate entropy metrics show that the random weight tests tend to be more like the original HITS tests.

The original HITS tests serve as the baseline for the number of ranked elements, as the results ranked by HITS-SW are always a subset. In figure 28, bars represent the number of elements in the *hi* (blue), *lo* (red), and median random (yellow) queries, as a fraction of the original HITS. The *lo* queries typically rank the most number of elements—and far more than the *hi* queries. The median random is typically fairly constant, containing around 85% of the elements in the original HITS ranking. Figures 29, 30, 31, 32, and 33 are all displayed in the same manner, with the data normalized to the median random query.

The distance the top 10 elements traveled is almost universally lower in the random queries than *hi* query. The *lo* query is typically similar to the randomized query, although there are some cases where the *lo* query is very close to zero. This will be addressed more in subsequent sections, but often the *lo* will have a distance value that is very small, as the query is close to the original HITS. Usually the random queries will also be quite close to the original HITS value, and have a very small distance. The reader should consult the actual aggregate entropy data in appendix A.

The rate of intersection in the top 10 and top 20 elements are typically quite high—higher than either the *hi* or *lo* queries. Usually the *lo* query has a higher rate of intersection as many of the *lo* queries typically are quite similar to the original HITS ranking.

As a result of the great variation and no distinct pattern, it is not possible to comment on the relation between the number of inversions in the randomized results and those in the *hi* and *lo* queries.

The rankings for all of the tested queries typically look like the original HITS output, occasionally with a few random elements. In many ways the HITS-SW *lo* queries typically

---

<sup>19</sup>There is the important caveat that our implementation of HITS-SW relies on cached keywords that are from December 2000—roughly four months older than the current pages on the Web.

Figure 28: HITS-SW Aggregate Entropy Comparison of Ranked Element Counts

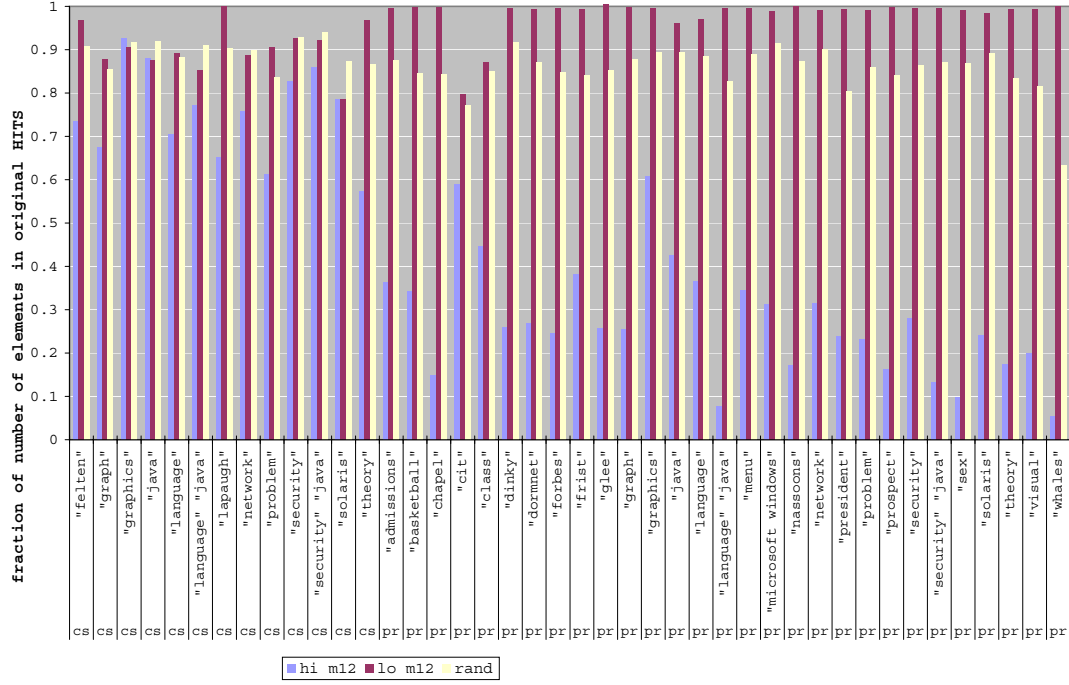


Figure 29: HITS-SW Aggregate Entropy Comparison of Distance

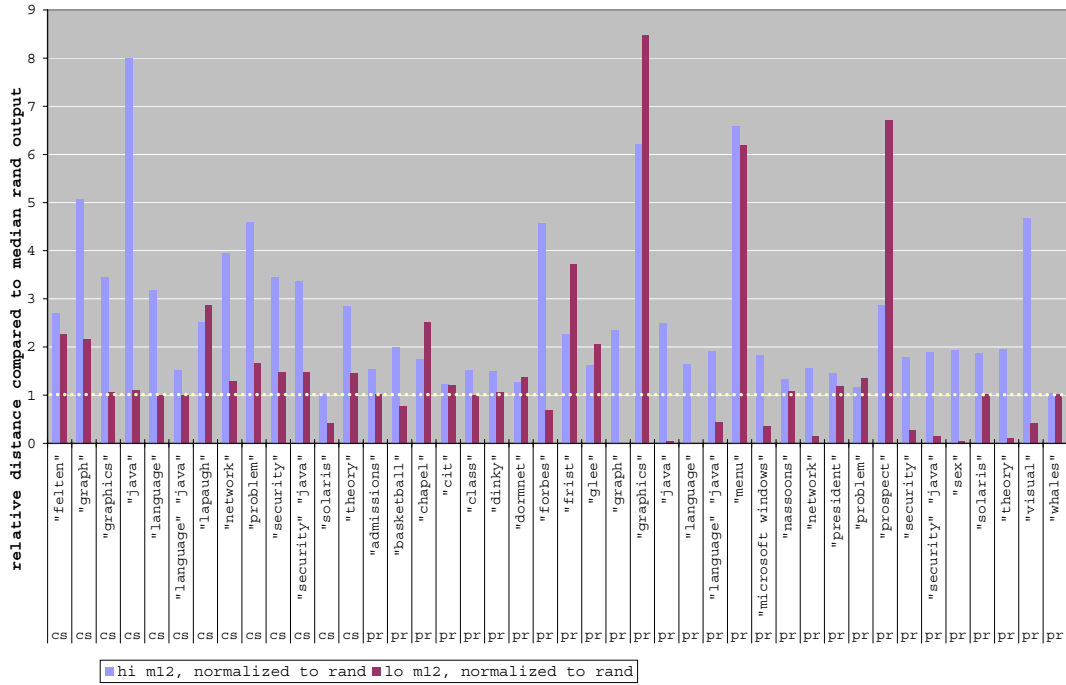


Figure 30: HITS-SW Aggregate Entropy Comparison of Intersection - *top10*

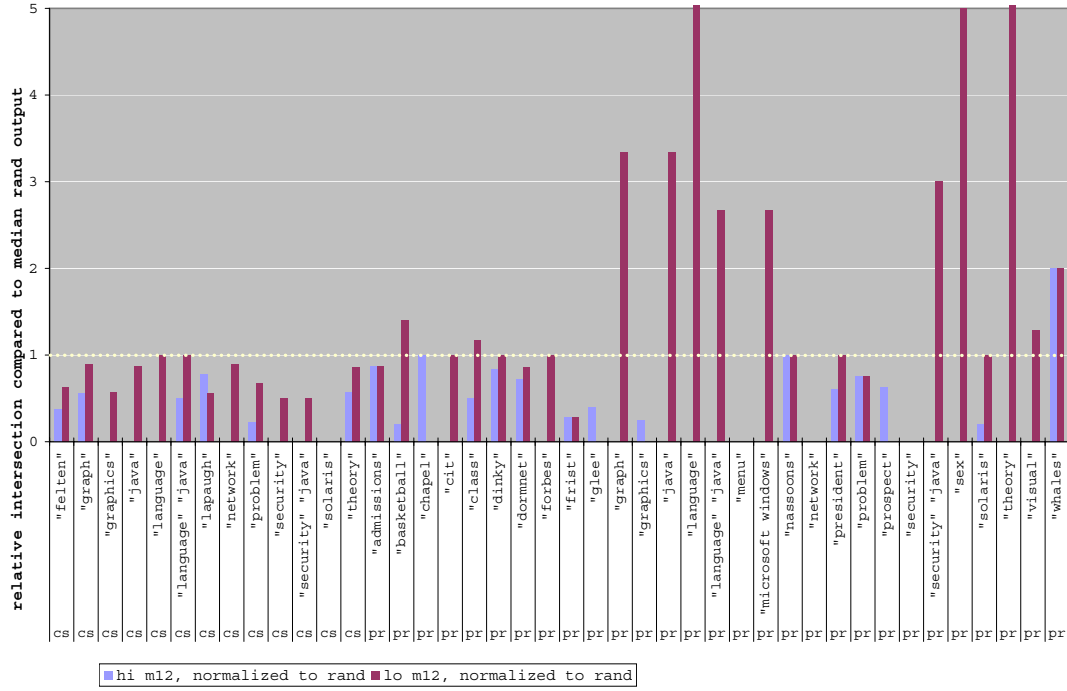


Figure 31: HITS-SW Aggregate Entropy Comparison of Intersection - *top20*

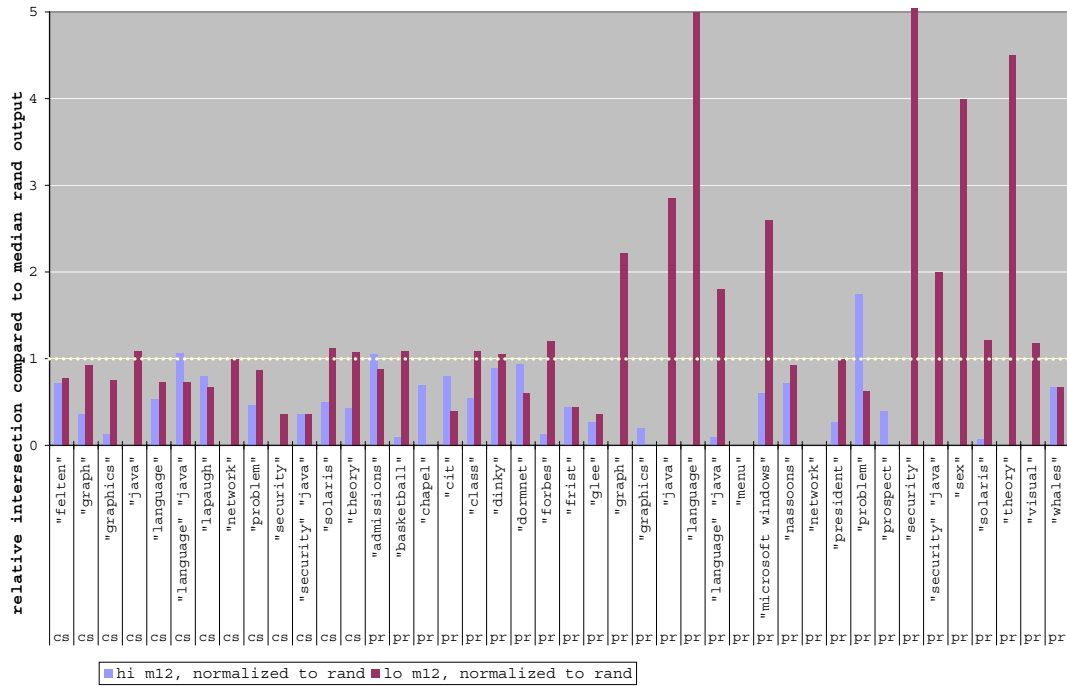


Figure 32: HITS-SW Aggregate Entropy Comparison of Inversions - *top10*

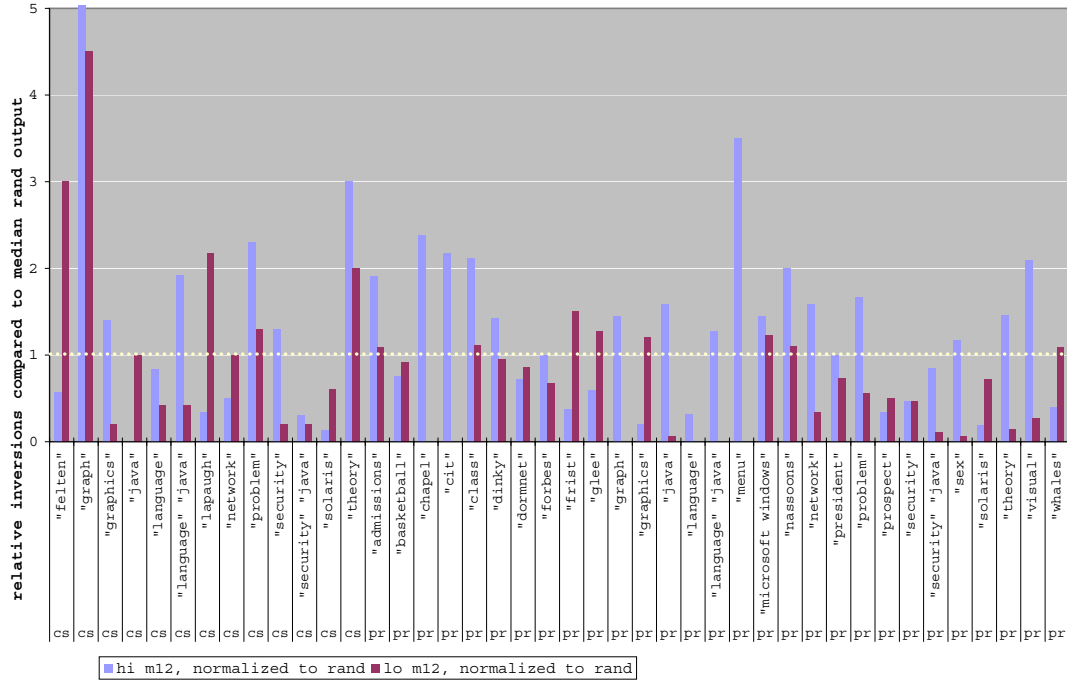
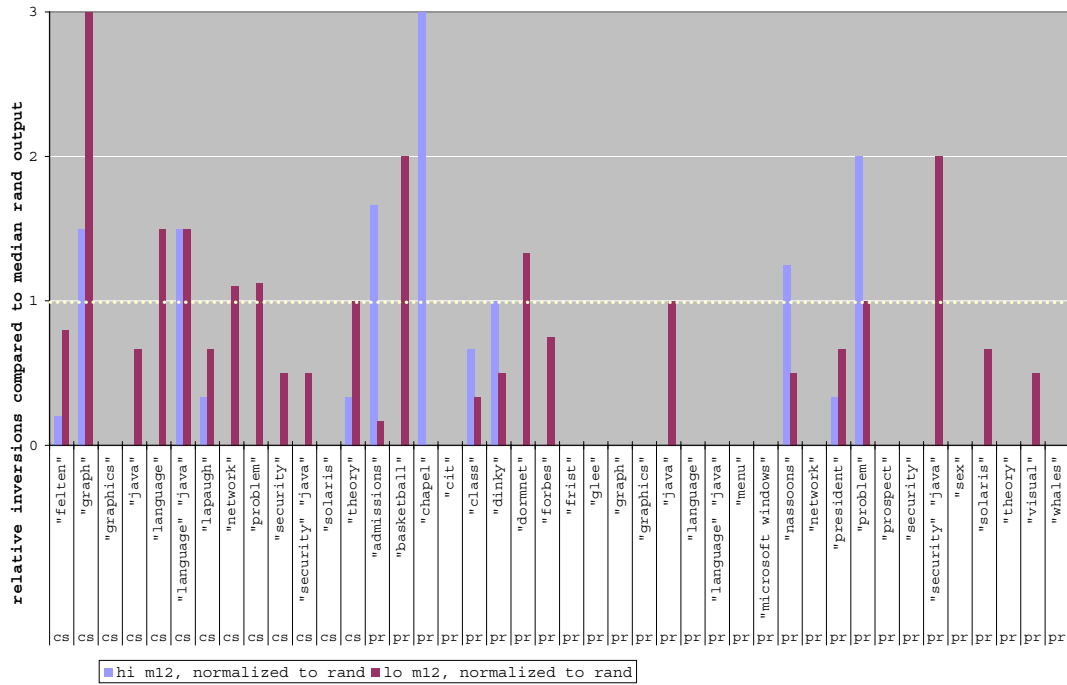


Figure 33: HITS-SW Aggregate Entropy Comparison of Inversions - *top10 original*



have values for the entropy measurements suggesting they are quite similar to the original HITS. The random weighting entropy data suggests that the random weighting rankings are even closer to the original HITS ranking. Observing the actual ranked output provides additional support for this claim.

## 6.2 Example HITS-SW Queries

### 6.2.1 “java” at Princeton

Figure 34 is the output from the standard HITS algorithm in a query for “java” on the Princeton domain. Notice that each page is on the “popindex” server. This site serves as a major archive for population research statistics. Part of this database contains an index for hundreds of research articles and citations. Each of the documents ranked in figure 34 contains the word java and are considered to be authoritative sources, and thus are highly ranked.

Figure 34: Ranking of HITS for “java” at Princeton

1	popindex.princeton.edu/browse/v55/n3/n.html
2	popindex.princeton.edu/browse/v53/n2/f.html
3	popindex.princeton.edu/browse/v52/n1/f.html
4	popindex.princeton.edu/browse/v53/n3/f.html
5	popindex.princeton.edu/browse/v55/n2/f.html
6	popindex.princeton.edu/browse/v57/n1/f.html
7	popindex.princeton.edu/browse/v53/n1/f.html
8	popindex.princeton.edu/browse/v55/n1/f.html
9	popindex.princeton.edu/browse/v52/n2/f.html
10	popindex.princeton.edu/browse/v52/n4/f.html
11	popindex.princeton.edu/browse/v58/n1/f.html
12	popindex.princeton.edu/browse/v57/n3/f.html
13	popindex.princeton.edu/browse/v56/n3/f.html
14	popindex.princeton.edu/browse/v58/n2/f.html
15	popindex.princeton.edu/browse/v56/n2/f.html
16	popindex.princeton.edu/browse/v58/n3/f.html
17	popindex.princeton.edu/browse/v62/n3/f.html
18	popindex.princeton.edu/browse/v59/n1/f.html
19	popindex.princeton.edu/browse/v54/n3/f.html
20	popindex.princeton.edu/browse/v59/n4/f.html

Figure 35 itemizes the top 20 results from running the *hi* query using the `mean_12` similarity weight data. The format of the figure describes the original rank in the first column and the new rank in the second column. Thus, the item now ranked as number 1 was formerly number 881. The pages located within “~matalive” are a part of the Math 199 Math Alive course home page and the pages located on the “storacle” server are Java API documentation.

Figure 36 enumerates the top 20 results from running the *lo* query with the `mean_12` similarity weighting. Again, the documents are all a part of the Population Index library. The results for running the same query, but with HITS-SW similarity parameter set as the range [60th, 100]<sup>20</sup> for the `mean_12` aggregate weights are located in figure 37. The pages

---

<sup>20</sup>all values greater than that corresponding to the 60th percentile

located within “~barshied” are a part of the personal homepage for an undergraduate student.

Figure 35: Ranking of HITS-SW *hi m12* for “java” at Princeton

884	1	<a href="http://www.princeton.edu/~matalive/VirtualClassroom/v0.1/html/lab2/lab2_6.html">www.princeton.edu/~matalive/VirtualClassroom/v0.1/html/lab2/lab2_6.html</a>
885	2	<a href="http://www.princeton.edu/~matalive/VirtualClassroom/v0.1/html/lab2/lab2_4.html">www.princeton.edu/~matalive/VirtualClassroom/v0.1/html/lab2/lab2_4.html</a>
886	3	<a href="http://www.princeton.edu/~matalive/VirtualClassroom/v0.1/html/lab2/lab2_3.html">www.princeton.edu/~matalive/VirtualClassroom/v0.1/html/lab2/lab2_3.html</a>
739	4	<a href="http://www.princeton.edu/~matalive/VirtualClassroom/v0.1/html/lab5/controls_1_1.html">www.princeton.edu/~matalive/VirtualClassroom/v0.1/html/lab5/controls_1_1.html</a>
738	5	<a href="http://www.princeton.edu/~matalive/VirtualClassroom/v0.1/html/lab5/controls_3_1.html">www.princeton.edu/~matalive/VirtualClassroom/v0.1/html/lab5/controls_3_1.html</a>
511	6	<a href="http://www.princeton.edu/Siteware/AnnArchiveApr1996.shtml">www.princeton.edu/Siteware/AnnArchiveApr1996.shtml</a>
761	7	<a href="http://www.princeton.edu/~matalive/VirtualClassroom/v0.1/html/lab5/controls_3_7.html">www.princeton.edu/~matalive/VirtualClassroom/v0.1/html/lab5/controls_3_7.html</a>
760	8	<a href="http://www.princeton.edu/~matalive/VirtualClassroom/v0.1/html/lab5/controls_3_5.html">www.princeton.edu/~matalive/VirtualClassroom/v0.1/html/lab5/controls_3_5.html</a>
759	9	<a href="http://www.princeton.edu/~matalive/VirtualClassroom/v0.1/html/lab5/controls_3_6.html">www.princeton.edu/~matalive/VirtualClassroom/v0.1/html/lab5/controls_3_6.html</a>
763	10	<a href="http://www.princeton.edu/~matalive/VirtualClassroom/v0.1/html/lab5/controls_3_3.html">www.princeton.edu/~matalive/VirtualClassroom/v0.1/html/lab5/controls_3_3.html</a>
762	11	<a href="http://www.princeton.edu/~matalive/VirtualClassroom/v0.1/html/lab5/controls_3_8.html">www.princeton.edu/~matalive/VirtualClassroom/v0.1/html/lab5/controls_3_8.html</a>
765	12	<a href="http://www.princeton.edu/~matalive/VirtualClassroom/v0.1/html/lab5/controls_3_9.html">www.princeton.edu/~matalive/VirtualClassroom/v0.1/html/lab5/controls_3_9.html</a>
766	13	<a href="http://www.princeton.edu/~matalive/VirtualClassroom/v0.1/html/lab5/controls_3_2.html">www.princeton.edu/~matalive/VirtualClassroom/v0.1/html/lab5/controls_3_2.html</a>
933	14	<a href="http://www.princeton.edu/~matalive/VirtualClassroom/v0.1/html/lab1/lab1_8_4.html">www.princeton.edu/~matalive/VirtualClassroom/v0.1/html/lab1/lab1_8_4.html</a>
093	15	<a href="http://storacle.princeton.edu:9001/ows-adoc/java/Package-java.lang.html">storacle.princeton.edu:9001/ows-adoc/java/Package-java.lang.html</a>
105	16	<a href="http://storacle.princeton.edu:9001/ows-adoc/java/java.lang.RuntimeException.html">storacle.princeton.edu:9001/ows-adoc/java/java.lang.RuntimeException.html</a>
106	17	<a href="http://storacle.princeton.edu:9001/ows-adoc/java/java.lang.Exception.html">storacle.princeton.edu:9001/ows-adoc/java/java.lang.Exception.html</a>
131	18	<a href="http://storacle.princeton.edu:9001/ows-adoc/java/java.lang.LinkageError.html">storacle.princeton.edu:9001/ows-adoc/java/java.lang.LinkageError.html</a>
145	19	<a href="http://storacle.princeton.edu:9001/ows-adoc/java/java.lang.VirtualMachineError.html">storacle.princeton.edu:9001/ows-adoc/java/java.lang.VirtualMachineError.html</a>
147	20	<a href="http://storacle.princeton.edu:9001/ows-adoc/java/java.lang.IncompatibleClassChangeError.html">storacle.princeton.edu:9001/ows-adoc/java/java.lang.IncompatibleClassChangeError.html</a>

### 6.2.2 “shrimp” at Computer Science

Figure 38 is the output from the standard HITS algorithm in a query for “shrimp” on the Computer Science domain. The intent is not to find prawns, but SHRIMP—Scalable High-performance Really Inexpensive Multi-Processor—the research project by the Computer Science department.<sup>21</sup>

Many of the ranked pages are in the shrimp subdirectory of the CS server. These pages are in the section devoted to the SHRIMP project. With the exception of the first document, the other ranked pages correspond to personal homepages of graduate students who have worked on the SHRIMP project. The first document is from the personal homepage of Mike Bostock, an undergraduate student, and contains a reference to shrimp, the food.

The total number of ranked results for this query is quite small, and thus when ranking items using HITS-SW, there are fewer than 20 items ranked. Figure 39 itemizes the top 18 of the results of running the HITS-SW *hi* query with *mean\_12* similarity weighting data.

Figure 40 enumerates the top 11 results from running the *lo* query with the *mean\_12* similarity weighting data. Figure 41 displays the results from the query using similarity weights greater than the 80th percentile.

As the “shrimp” query results in a small number of documents, it is not reasonable to examine the standard randomized control test data. In such a small set of pages, examining 25 random permutations would cover most of the universe of possibilities and not provide as much value as if the query returned hundreds of results.

<sup>21</sup>Although the acronym is uppercase, this query was run as if the user queried for the lowercase “shrimp”.

Figure 36: Ranking of HITS-SW *lo m12* for “java” at Princeton

1	1	popindex.princeton.edu/browse/v55/n3/n.html
2	2	popindex.princeton.edu/browse/v53/n2/f.html
3	3	popindex.princeton.edu/browse/v52/n1/f.html
4	4	popindex.princeton.edu/browse/v53/n3/f.html
5	5	popindex.princeton.edu/browse/v55/n2/f.html
6	6	popindex.princeton.edu/browse/v57/n1/f.html
7	7	popindex.princeton.edu/browse/v53/n1/f.html
9	8	popindex.princeton.edu/browse/v52/n2/f.html
8	9	popindex.princeton.edu/browse/v55/n1/f.html
10	10	popindex.princeton.edu/browse/v52/n4/f.html
11	11	popindex.princeton.edu/browse/v58/n1/f.html
12	12	popindex.princeton.edu/browse/v57/n3/f.html
13	13	popindex.princeton.edu/browse/v56/n3/f.html
14	14	popindex.princeton.edu/browse/v58/n2/f.html
15	15	popindex.princeton.edu/browse/v56/n2/f.html
16	16	popindex.princeton.edu/browse/v58/n3/f.html
18	17	popindex.princeton.edu/browse/v59/n1/f.html
17	18	popindex.princeton.edu/browse/v62/n3/f.html
19	19	popindex.princeton.edu/browse/v54/n3/f.html
20	20	popindex.princeton.edu/browse/v59/n4/f.html

Figure 37: Ranking of HITS *60up m12* for “java” at Princeton

538	1	www.princeton.edu/~barshied/planets/planets.html
536	2	www.princeton.edu/~barshied/teletubbies/teletubbies.html
535	3	www.princeton.edu/~barshied/live/live.html
534	4	www.princeton.edu/~barshied/past/class98.html
533	5	www.princeton.edu/~barshied/guest/guestbook.htm
532	6	www.princeton.edu/~barshied/sounds/dmbtapelist.html
531	7	www.princeton.edu/~barshied/mand/mand.html
539	8	www.princeton.edu/~barshied/java/javagame.html
537	9	www.princeton.edu/~barshied/dragon/dragon.html
637	10	www.princeton.edu/~barshied/java/play.html
768	11	www.princeton.edu/pr/pwb/00/0508/p/empl.shtml
525	12	xray5.princeton.edu/~xin/program.html
526	13	xray2.princeton.edu/~xin/program.html
633	14	infoshare1.princeton.edu/katmandu/marc/043b.html
769	15	libweb.princeton.edu/katmandu/marc/043b.html
505	16	www.princeton.edu/~as/ProgrammingServices.html
933	17	www.princeton.edu/~matalive/VirtualClassroom/v0.1/html/lab1/lab1_8_4.html
508	18	www.princeton.edu/as/ProgrammingServices.html
885	19	www.princeton.edu/~matalive/VirtualClassroom/v0.1/html/lab2/lab2_4.html
428	20	storacle.princeton.edu:9001/ows-adoc/java/oracle.plsql.PLSQLRuntimeException.html

Figure 38: Ranking of HITS for “shrimp” at Computer Science

1	<a href="http://www.cs.princeton.edu/~mbostock/locale/princeton_old.html">www.cs.princeton.edu/~mbostock/locale/princeton_old.html</a>
2	<a href="http://www.cs.princeton.edu/shrimp/html/platforms.html">www.cs.princeton.edu/shrimp/html/platforms.html</a>
3	<a href="http://www.cs.princeton.edu/shrimp/html/papers.html">www.cs.princeton.edu/shrimp/html/papers.html</a>
4	<a href="http://www.cs.princeton.edu/~scott/">www.cs.princeton.edu/~scott/</a>
5	<a href="http://www.cs.princeton.edu/~yzhou/">www.cs.princeton.edu/~yzhou/</a>
6	<a href="http://www.cs.princeton.edu/shrimp/html/sockets.html">www.cs.princeton.edu/shrimp/html/sockets.html</a>
7	<a href="http://www.cs.princeton.edu/shrimp/html/rpc.html">www.cs.princeton.edu/shrimp/html/rpc.html</a>
8	<a href="http://www.cs.princeton.edu/shrimp/html/nx.html">www.cs.princeton.edu/shrimp/html/nx.html</a>
9	<a href="http://www.cs.princeton.edu/shrimp/html/message_passing1.html">www.cs.princeton.edu/shrimp/html/message_passing1.html</a>
10	<a href="http://www.cs.princeton.edu/shrimp/html/myrinet.html">www.cs.princeton.edu/shrimp/html/myrinet.html</a>
11	<a href="http://www.cs.princeton.edu/shrimp/html/shrimp-ii.html">www.cs.princeton.edu/shrimp/html/shrimp-ii.html</a>
12	<a href="http://www.cs.princeton.edu/~cliao/">www.cs.princeton.edu/~cliao/</a>
13	<a href="http://www.cs.princeton.edu/~rda/">www.cs.princeton.edu/~rda/</a>
14	<a href="http://www.cs.princeton.edu/shrimp/html/graphics.html">www.cs.princeton.edu/shrimp/html/graphics.html</a>
15	<a href="http://www.cs.princeton.edu/shrimp/software/vmmc_nt_2_0.html">www.cs.princeton.edu/shrimp/software/vmmc_nt_2_0.html</a>
16	<a href="http://www.cs.princeton.edu/~cliao/research.html">www.cs.princeton.edu/~cliao/research.html</a>
17	<a href="http://www.cs.princeton.edu/~cliao/resume.html">www.cs.princeton.edu/~cliao/resume.html</a>
18	<a href="http://www.cs.princeton.edu/~mb/resume.html">www.cs.princeton.edu/~mb/resume.html</a>
19	<a href="http://www.cs.princeton.edu/~mb/bookm.html">www.cs.princeton.edu/~mb/bookm.html</a>
20	<a href="http://www.cs.princeton.edu/gifts/mentor/">www.cs.princeton.edu/gifts/mentor/</a>

Figure 39: Ranking of HITS-SW *hi m12* for “shrimp” at Computer Science

1	1	<a href="http://www.cs.princeton.edu/~mbostock/locale/princeton_old.html">www.cs.princeton.edu/~mbostock/locale/princeton_old.html</a>
3	2	<a href="http://www.cs.princeton.edu/shrimp/html/papers.html">www.cs.princeton.edu/shrimp/html/papers.html</a>
2	3	<a href="http://www.cs.princeton.edu/shrimp/html/platforms.html">www.cs.princeton.edu/shrimp/html/platforms.html</a>
6	4	<a href="http://www.cs.princeton.edu/shrimp/html/sockets.html">www.cs.princeton.edu/shrimp/html/sockets.html</a>
7	5	<a href="http://www.cs.princeton.edu/shrimp/html/rpc.html">www.cs.princeton.edu/shrimp/html/rpc.html</a>
8	6	<a href="http://www.cs.princeton.edu/shrimp/html/nx.html">www.cs.princeton.edu/shrimp/html/nx.html</a>
11	7	<a href="http://www.cs.princeton.edu/shrimp/html/shrimp-ii.html">www.cs.princeton.edu/shrimp/html/shrimp-ii.html</a>
9	8	<a href="http://www.cs.princeton.edu/shrimp/html/message_passing1.html">www.cs.princeton.edu/shrimp/html/message_passing1.html</a>
10	9	<a href="http://www.cs.princeton.edu/shrimp/html/myrinet.html">www.cs.princeton.edu/shrimp/html/myrinet.html</a>
14	10	<a href="http://www.cs.princeton.edu/shrimp/html/graphics.html">www.cs.princeton.edu/shrimp/html/graphics.html</a>
12	11	<a href="http://www.cs.princeton.edu/~cliao/">www.cs.princeton.edu/~cliao/</a>
16	12	<a href="http://www.cs.princeton.edu/~cliao/research.html">www.cs.princeton.edu/~cliao/research.html</a>
17	13	<a href="http://www.cs.princeton.edu/~cliao/resume.html">www.cs.princeton.edu/~cliao/resume.html</a>
13	14	<a href="http://www.cs.princeton.edu/~rda/">www.cs.princeton.edu/~rda/</a>
4	15	<a href="http://www.cs.princeton.edu/~scott/">www.cs.princeton.edu/~scott/</a>
20	16	<a href="http://www.cs.princeton.edu/gifts/mentor/">www.cs.princeton.edu/gifts/mentor/</a>
23	17	<a href="http://www.cs.princeton.edu/shrimp/html/shrimp-iii.html">www.cs.princeton.edu/shrimp/html/shrimp-iii.html</a>
5	18	<a href="http://www.cs.princeton.edu/~yzhou/">www.cs.princeton.edu/~yzhou/</a>

Figure 40: Ranking of HITS-SW *lo m12* for “shrimp” at Computer Science

4	1	<a href="http://www.cs.princeton.edu/~scott/">www.cs.princeton.edu/~scott/</a>
5	2	<a href="http://www.cs.princeton.edu/~yzhou/">www.cs.princeton.edu/~yzhou/</a>
12	3	<a href="http://www.cs.princeton.edu/~cliao/">www.cs.princeton.edu/~cliao/</a>
13	4	<a href="http://www.cs.princeton.edu/~rda/">www.cs.princeton.edu/~rda/</a>
18	5	<a href="http://www.cs.princeton.edu/~mb/resume.html">www.cs.princeton.edu/~mb/resume.html</a>
21	6	<a href="http://www.cs.princeton.edu/grad/gradguide/integrat.htm">www.cs.princeton.edu/grad/gradguide/integrat.htm</a>
19	7	<a href="http://www.cs.princeton.edu/~mb/bookm.html">www.cs.princeton.edu/~mb/bookm.html</a>
15	8	<a href="http://www.cs.princeton.edu/shrimp/software/vmmc_nt_2_0.html">www.cs.princeton.edu/shrimp/software/vmmc_nt_2_0.html</a>
1	9	<a href="http://www.cs.princeton.edu/~mbostock/locale/princeton_old.html">www.cs.princeton.edu/~mbostock/locale/princeton_old.html</a>
22	10	<a href="http://www.cs.princeton.edu/~snd/coolstuff.html">www.cs.princeton.edu/~snd/coolstuff.html</a>
23	11	<a href="http://www.cs.princeton.edu/shrimp/html/shrimp-ii1.html">www.cs.princeton.edu/shrimp/html/shrimp-ii1.html</a>

Figure 41: Ranking of HITS-SW *80up m12* for “shrimp” at Computer Science

16	1	<a href="http://www.cs.princeton.edu/~cliao/research.html">www.cs.princeton.edu/~cliao/research.html</a>
17	2	<a href="http://www.cs.princeton.edu/~cliao/resume.html">www.cs.princeton.edu/~cliao/resume.html</a>
12	3	<a href="http://www.cs.princeton.edu/~cliao/">www.cs.princeton.edu/~cliao/</a>

### 6.2.3 “sex” at Princeton

Figure 42 is the output from the standard HITS algorithm in a query for “sex” on the Princeton domain. Figure 43 lists the top 20 results from running the *hi* query using the `mean_u` similarity weight data; the majority of the ranked pages come from the Nassau Weekly publication website. Figure 44 lists the top 20 results from running the *hi* query using the `mean_12` similarity data.

Figure 45 enumerates the top 20 results from running the *lo* query using the `mean_u` similarity weight data while figure 46 is the *lo* query using the `mean_12` data. The pages from the *mu* query are a part of a Tcl documentation site.

## 6.3 Analysis of HITS-SW Experiments

### 6.3.1 HITS as similar to HITS-SW *lo*

Reexamine the “java” query in section 6.2.1. The reader should note that the results for the *hi* query are very different from the original HITS ranking, while the results for the *lo* query are almost identical to the original version. This pattern is seen in about a fifth of the queries run, and mostly only within the Princeton domain.

This would indicate that the original HITS query could be characterized as one that was primarily dominated by low-similarity hyperlinks. In fact, 23 of the 25 random results look like the original ranking but are composed of *other* pages from the popindex server. The data is quite compelling.

These highly ranked clustered pages contain the relevant query terms (“java”). Presumably many of these documents are considered authoritative because of the large number of links used for navigation in such a large index. There are, of course, a number of legitimate

Figure 42: Ranking of HITS for “sex” at Princeton

```

1  popindex.princeton.edu/browse/v59/n4/ai.html
2  popindex.princeton.edu/browse/v59/n2/n.html
3  popindex.princeton.edu/browse/v59/n1/n.html
4  popindex.princeton.edu/browse/v62/n3/n.html
5  popindex.princeton.edu/browse/v65/n4/n.html
6  popindex.princeton.edu/browse/v65/n2/n.html
7  popindex.princeton.edu/browse/v57/n3/n.html
8  popindex.princeton.edu/browse/v57/n1/n.html
9  popindex.princeton.edu/browse/v62/n2/n.html
10 popindex.princeton.edu/browse/v52/n3/n.html
11 popindex.princeton.edu/browse/v54/n2/n.html
12 popindex.princeton.edu/browse/v54/n1/n.html
13 popindex.princeton.edu/browse/v55/n2/n.html
14 popindex.princeton.edu/browse/v55/n3/n.html
15 popindex.princeton.edu/browse/v64/n2/n.html
16 popindex.princeton.edu/browse/v58/n2/n.html
17 popindex.princeton.edu/browse/v58/n1/n.html
18 popindex.princeton.edu/browse/v54/n4/n.html
19 popindex.princeton.edu/browse/v56/n3/n.html
20 popindex.princeton.edu/browse/v58/n3/n.html

```

Figure 43: Ranking of HITS-SW *hi mu* for “sex” at Princeton

```

762 1  www.princeton.edu/~paw/order_100years.html
763 2  www.princeton.edu/~nweekly/forum/index.html
765 3  www.princeton.edu/~nweekly/culture/1999/11/stern/index.html
766 4  www.princeton.edu/~nweekly/forum/1999/11/bolivia/index.html
767 5  www.princeton.edu/~nweekly/culture/2000/02/beachrev/index.html
768 6  www.princeton.edu/~nweekly/culture/2000/02/jeffreyrev/index.html
770 7  www.princeton.edu/~nweekly/fiction/1999/12/benson/index.html
774 8  www.princeton.edu/~nweekly/culture/1999/11/dismemberment/index.html
773 9  www.princeton.edu/~nweekly/culture/1999/11/rage/index.html
772 10 www.princeton.edu/~nweekly/page2/1999/10/frenchfry/index.html
776 11 www.princeton.edu/~nweekly/forum/2000/01/ethnic/index.html
764 12 www.princeton.edu/~nweekly/verbatim/1999/09/30/index.html
777 13 www.princeton.edu/~nweekly/culture/1999/10/whitney/index.html
778 14 www.princeton.edu/~nweekly/forum/1999/09/mensmags/index.html
779 15 www.princeton.edu/~nweekly/news/1999/10/conspiracy/index.html
780 16 www.princeton.edu/~nweekly/news/2000/02/fakecandidacy/index.html
781 17 www.princeton.edu/~nweekly/news/1999/11/mahir/index.html
782 18 www.princeton.edu/~nweekly/forum/2000/02/handjob2/index.html
769 19 www.princeton.edu/~nweekly/verbatim/1999/11/11/index.html
787 20 www.princeton.edu/~nweekly/forum/2000/02/hospital/index.html

```

Figure 44: Ranking of HITS-SW *hi m12* for “sex” at Princeton

1059	1	<a href="http://www.princeton.edu/pr/pwb/00/0910/7c.shtml">www.princeton.edu/pr/pwb/00/0910/7c.shtml</a>
834	2	<a href="http://www.dailyprincetonian.com/Content/2000/05/05/edits/196.html">www.dailyprincetonian.com/Content/2000/05/05/edits/196.html</a>
1261	3	<a href="http://www.princeton.edu/pr/pwb/00/0925/2a.shtml">www.princeton.edu/pr/pwb/00/0925/2a.shtml</a>
833	4	<a href="http://www.dailyprincetonian.com/Content/2000/04/25/news/335.html">www.dailyprincetonian.com/Content/2000/04/25/news/335.html</a>
863	5	<a href="http://www.dailyprincetonian.com/Content/2000/09/20/edits/235.html">www.dailyprincetonian.com/Content/2000/09/20/edits/235.html</a>
832	6	<a href="http://www.dailyprincetonian.com/Content/2000/09/27/edits/246.html">www.dailyprincetonian.com/Content/2000/09/27/edits/246.html</a>
868	7	<a href="http://www.dailyprincetonian.com/Content/2000/09/27/news/538.html">www.dailyprincetonian.com/Content/2000/09/27/news/538.html</a>
867	8	<a href="http://www.dailyprincetonian.com/Content/2000/09/25/news/527.html">www.dailyprincetonian.com/Content/2000/09/25/news/527.html</a>
866	9	<a href="http://www.dailyprincetonian.com/Content/2000/09/20/news/500.html">www.dailyprincetonian.com/Content/2000/09/20/news/500.html</a>
1110	10	<a href="http://www.princeton.edu/~humanres/ben/publicat.htm">www.princeton.edu/~humanres/ben/publicat.htm</a>
1047	11	<a href="http://www.princeton.edu/hr/ben/publicat.htm">www.princeton.edu/hr/ben/publicat.htm</a>
851	12	<a href="http://www.princeton.edu/~lgba/Archives/Pride/Pride97/thursday.shtml">www.princeton.edu/~lgba/Archives/Pride/Pride97/thursday.shtml</a>
1315	13	<a href="http://www.princeton.edu/~humanres/ben/domtoc.htm">www.princeton.edu/~humanres/ben/domtoc.htm</a>
1279	14	<a href="http://www.princeton.edu/hr/ben/domtoc.htm">www.princeton.edu/hr/ben/domtoc.htm</a>
1245	15	<a href="http://www.princeton.edu/pr/pwb/00/1002/3a.shtml">www.princeton.edu/pr/pwb/00/1002/3a.shtml</a>
1235	16	<a href="http://www.princeton.edu/~humanres/ben/children.htm">www.princeton.edu/~humanres/ben/children.htm</a>
1231	17	<a href="http://www.princeton.edu/hr/ben/children.htm">www.princeton.edu/hr/ben/children.htm</a>
775	18	<a href="http://www.princeton.edu/~nweekly/verbatim/1999/12/02/index.html">www.princeton.edu/~nweekly/verbatim/1999/12/02/index.html</a>
771	19	<a href="http://www.princeton.edu/~nweekly/verbatim/1999/11/18/index.html">www.princeton.edu/~nweekly/verbatim/1999/11/18/index.html</a>
769	20	<a href="http://www.princeton.edu/~nweekly/verbatim/1999/11/11/index.html">www.princeton.edu/~nweekly/verbatim/1999/11/11/index.html</a>

Figure 45: Ranking of HITS-SW *lo mu* for “sex” at Princeton

1	1	<a href="http://popindex.princeton.edu/browse/v59/n4/ai.html">popindex.princeton.edu/browse/v59/n4/ai.html</a>
142	2	<a href="http://popindex.princeton.edu/browse/v59/n3/h.html">popindex.princeton.edu/browse/v59/n3/h.html</a>
242	3	<a href="http://popindex.princeton.edu/browse/v59/n3/o.html">popindex.princeton.edu/browse/v59/n3/o.html</a>
140	4	<a href="http://popindex.princeton.edu/browse/v59/n3/m.html">popindex.princeton.edu/browse/v59/n3/m.html</a>
236	5	<a href="http://popindex.princeton.edu/browse/v59/n3/l.html">popindex.princeton.edu/browse/v59/n3/l.html</a>
377	6	<a href="http://popindex.princeton.edu/browse/v59/n3/k.html">popindex.princeton.edu/browse/v59/n3/k.html</a>
233	7	<a href="http://popindex.princeton.edu/browse/v59/n3/g.html">popindex.princeton.edu/browse/v59/n3/g.html</a>
373	8	<a href="http://popindex.princeton.edu/browse/v59/n3/e.html">popindex.princeton.edu/browse/v59/n3/e.html</a>
375	9	<a href="http://popindex.princeton.edu/browse/v59/n3/d.html">popindex.princeton.edu/browse/v59/n3/d.html</a>
392	10	<a href="http://popindex.princeton.edu/browse/v59/n3/c.html">popindex.princeton.edu/browse/v59/n3/c.html</a>
385	11	<a href="http://popindex.princeton.edu/browse/v59/n3/b.html">popindex.princeton.edu/browse/v59/n3/b.html</a>
84	12	<a href="http://popindex.princeton.edu/browse/v59/n4/s.html">popindex.princeton.edu/browse/v59/n4/s.html</a>
243	13	<a href="http://popindex.princeton.edu/browse/v59/n4/o.html">popindex.princeton.edu/browse/v59/n4/o.html</a>
141	14	<a href="http://popindex.princeton.edu/browse/v59/n4/m.html">popindex.princeton.edu/browse/v59/n4/m.html</a>
235	15	<a href="http://popindex.princeton.edu/browse/v59/n4/l.html">popindex.princeton.edu/browse/v59/n4/l.html</a>
143	16	<a href="http://popindex.princeton.edu/browse/v59/n4/h.html">popindex.princeton.edu/browse/v59/n4/h.html</a>
234	17	<a href="http://popindex.princeton.edu/browse/v59/n4/g.html">popindex.princeton.edu/browse/v59/n4/g.html</a>
374	18	<a href="http://popindex.princeton.edu/browse/v59/n4/e.html">popindex.princeton.edu/browse/v59/n4/e.html</a>
391	19	<a href="http://popindex.princeton.edu/browse/v59/n4/c.html">popindex.princeton.edu/browse/v59/n4/c.html</a>
376	20	<a href="http://popindex.princeton.edu/browse/v59/n4/d.html">popindex.princeton.edu/browse/v59/n4/d.html</a>

Figure 46: Ranking of HITS-SW *lo m12* for “sex” at Princeton

1	1	<a href="http://popindex.princeton.edu/browse/v59/n4/ai.html">popindex.princeton.edu/browse/v59/n4/ai.html</a>
2	2	<a href="http://popindex.princeton.edu/browse/v59/n2/n.html">popindex.princeton.edu/browse/v59/n2/n.html</a>
3	3	<a href="http://popindex.princeton.edu/browse/v59/n1/n.html">popindex.princeton.edu/browse/v59/n1/n.html</a>
4	4	<a href="http://popindex.princeton.edu/browse/v62/n3/n.html">popindex.princeton.edu/browse/v62/n3/n.html</a>
5	5	<a href="http://popindex.princeton.edu/browse/v65/n4/n.html">popindex.princeton.edu/browse/v65/n4/n.html</a>
6	6	<a href="http://popindex.princeton.edu/browse/v65/n2/n.html">popindex.princeton.edu/browse/v65/n2/n.html</a>
8	7	<a href="http://popindex.princeton.edu/browse/v57/n1/n.html">popindex.princeton.edu/browse/v57/n1/n.html</a>
7	8	<a href="http://popindex.princeton.edu/browse/v57/n3/n.html">popindex.princeton.edu/browse/v57/n3/n.html</a>
9	9	<a href="http://popindex.princeton.edu/browse/v62/n2/n.html">popindex.princeton.edu/browse/v62/n2/n.html</a>
10	10	<a href="http://popindex.princeton.edu/browse/v52/n3/n.html">popindex.princeton.edu/browse/v52/n3/n.html</a>
12	11	<a href="http://popindex.princeton.edu/browse/v54/n1/n.html">popindex.princeton.edu/browse/v54/n1/n.html</a>
11	12	<a href="http://popindex.princeton.edu/browse/v54/n2/n.html">popindex.princeton.edu/browse/v54/n2/n.html</a>
13	13	<a href="http://popindex.princeton.edu/browse/v55/n2/n.html">popindex.princeton.edu/browse/v55/n2/n.html</a>
14	14	<a href="http://popindex.princeton.edu/browse/v55/n3/n.html">popindex.princeton.edu/browse/v55/n3/n.html</a>
15	15	<a href="http://popindex.princeton.edu/browse/v64/n2/n.html">popindex.princeton.edu/browse/v64/n2/n.html</a>
16	16	<a href="http://popindex.princeton.edu/browse/v58/n2/n.html">popindex.princeton.edu/browse/v58/n2/n.html</a>
17	17	<a href="http://popindex.princeton.edu/browse/v58/n1/n.html">popindex.princeton.edu/browse/v58/n1/n.html</a>
18	18	<a href="http://popindex.princeton.edu/browse/v54/n4/n.html">popindex.princeton.edu/browse/v54/n4/n.html</a>
19	19	<a href="http://popindex.princeton.edu/browse/v56/n3/n.html">popindex.princeton.edu/browse/v56/n3/n.html</a>
20	20	<a href="http://popindex.princeton.edu/browse/v58/n3/n.html">popindex.princeton.edu/browse/v58/n3/n.html</a>

citation links in these pages.<sup>22</sup>

### 6.3.2 Clustering in HITS and HITS-SW

Again consider the “java” query in section 6.2.1. The original HITS ranking does not provide the most important results first. Instead, the initial set of results are all from the same location and generally appear to be the same sort of results. This might be improved by applying current research on clustering to eliminate or to condense a portion of these pages, as the typical user would most likely not want all of his results to be from the same area of the Web.

Indeed, data from many queries (including the “sex” query presented in section 6.2.3) produced similar sorts of results, where the majority, if not all, of the ranked items came from one cluster: at least a quarter of the queries exhibited similar findings. In the Princeton domain, patterns emerged in such clusters. Many clusterings came from the popindex pages, but often enough others would be information included in courses; typically, these in courses were clusters of Java API documentation.

Knowing that often the *lo* queries would be dominated by clusters of data, one might hope that there would be no clustering in the *hi* queries. Because clustering appeared with great frequency in the *lo* query, we hoped that this pattern would be exclusive for this group. It was not: the *hi* query indeed brought in totally new elements similar to one another.<sup>23</sup> The results, however, were still rather clustered. Raising the similarity weight threshold did not help either: observe figure 37 where HITS-SW was run using weights above the 60th percentile.

<sup>22</sup>Although this is meant in a connectivity-analysis sense, in this query the links are *literally* citations!

<sup>23</sup>The fact that they were similar should not be a surprise, as the *hi* query utilized only those links between similar pages.

Although the majority of the Math Alive pages found in the *hi* query (figure 35) are eliminated, a cluster from a personal homepages surfaces in its place and the number of ranked results drops precipitously. The drop in ranked results should not surprise the reader; when we consider fewer edges—by narrowing the range of edge used as a parameter in HITS-SW—the number of edges used in the connectivity-analysis algorithm decreases. With fewer edges, the resulting graph is more sparse, which leads to additional nodes becoming isolated from other parts of the graph. Isolated nodes cannot receive high authority scores because they do not have adjacent edges and nodes to increase their authority values.

It is interesting that clustering is still noticeable even with the high similarity queries. Evidently, there are clusters of highly similar documents. But the cases that we have seen thus far are unexpected. The documents with the cluster do not appear to be similar, and yet the links within the cluster receive high similarity weights. These findings are attributed to the structural similarity of the documents.

### 6.3.3 Analysis of a small query

Returning to consider the query for “shrimp” in the CS domain, illustrated in section 6.2.2, one sees that the number of pages in the CS domain is much smaller than those throughout Princeton. With slightly more than 20 results ranked by HITS, it is acceptable to analyze the HITS-SW queries. The random-weighting tests do not provide any value given that with more random-weighting tests than elements in the set, these random tests will likely cover the universe of possible permutations.

The small size of this set allows us to focus specifically on the differences in ranking and the content and links of the original pages. While this analysis can provide a different level of insight, it becomes very subjective and is difficult to generalize to other queries.

In the *hi* query, the SHRIMP pages are promoted and the homepages are demoted. The opposite is the trend in the *lo* query, where the homepages are promoted. This does not seem surprising considering the content of each page: the homepages typically suggest that the owner has participated in the SHRIMP research project, yet content is otherwise dissimilar from all adjoining pages. The SHRIMP pages, however, have some content that is similar to the surrounding SHRIMP project pages.

Also interesting is the Bostock home page that remains highly ranked even when using the *hi* query. His page is not similar in content to the adjoining pages in the collection, but to prevent it from appearing, the similarity parameter must be changed to include very high similarity weights. Only a query using links weighted above the 80th percentile prevents the Bostock page from appearing. Though if you live by the sword, you die by the sword: such weight restrictions isolate all nodes in the graph so the number of ranked documents is very small. This is compounded by the small size of the initial collection.

### 6.3.4 Structural Similarity vs. Content Similarity

An interesting trend appears in both of the *hi* queries explained above. In the query for “java”, the *hi* results mostly consisted of pages for the Math Alive course. “Shrimp” included many SHRIMP pages and a page from the Bostock homepage. In all of these cases, the pages were deemed similar to the surrounding pages by the vector similarity approach used for HITS-SW. It is sometimes difficult to understand why certain pages are considered similar. This analysis will focus upon the Bostock page, but this reasoning applies to other documents as well.

The Bostock page is a part of a cluster of interlinked pages. Under the normal version of HITS, they tend to be promoted because there are numerous links between pages that provide for navigation, as suggested in the original HITS paper by Kleinberg, subsequent work by Baccash, and additional research Bharat and Henzinger. [20] [7] [8] Recall that Kleinberg’s approach is to ignore links within a single server. Since the data from the CS domain resides on one server, such elimination was impossible, short of implementing an arbitrary policy analogue for *directories* or specific paths. Because even pages within a cluster that do not appear to be similar receive high similarity weights, using a high threshold in HITS-SW does not seem to resolve this issue. If only to understand why such clusters were receiving high similarity weights, a notion of “structural similarity” developed.

Consider pages from two independent sites: *yahoo.com* and *yahoo.co.uk*. Both are English-language Yahoo webpages. Neither have links in common, and the actual content may be very different. (Assume that we have a page about government on the American Yahoo and a page about hoof-and-mouth disease from the British Yahoo.) Although these pages might have no actual common content, they are structurally similar: both have heading information that can be identified as Yahoo and common category titles and information. A person can visually identify these sites as similar, and this judgment is based upon more than graphical elements, colors, and layout in common: it is based the similarity in actual content. A person can easily separate the Yahoo look and feel from the content that adds value—the category information; this task is more difficult for computers.

Because every recognizable piece of text is turned into a keyword, no distinction is made between navigational content and the ‘real’ content. The Bostock page that contained the word “shrimp” links to other pages in his homepage site. All had the same look and feel, and often the words lending to the design had a significant impact on the total number of keywords found in the documents. This was also the case on the SHRIMP pages; all of the pages in this cluster were from a FrontPage HTML template. All contained the same sort of links and the associated anchor text. Because we used this anchor text as a source of keywords, we inadvertently biased our similarity metric to include navigational *text*.

All three queries presented in this paper serve as simple examples of this structural similarity effect, but our other queries exhibit this effect, too.

### 6.3.5 Using Links for the Similarity Measurement in HITS-SW

The reasoning behind using links as “terms” for similarity weighting was that they might produce interesting results and that such a modification was simple given the infrastructure already implemented. The hypothesis was that the similarity weights would be very different from the content-based similarity weights. (See section 4.2.1 for more information on the similarity weighting mechanisms used in HITS-SW.)

Indeed, the results for the HITS-SW queries using the *mu* data are often dissimilar from the respective queries using the *m12* data. Examine the HITS-SW *hi* queries for “sex” using similarity data *m12* and *mu* in figures 43 and 44. The ranking based upon *mu* is highly clustered whereas the ranking for *m12* is not. This might lead to the hypothesis that using links in common as a similarity metric leads to clustering. The data does not support such a hypothesis, however, as the *lo* “sex” query using *mu* is clustered as well. There is no noticeable pattern to clustering in HITS-SW using *mu* data; as explained in section 6.3.2, clustering is often a problem in *m12 hi* and *lo* queries.

## 7 Conclusions

### Or, Where will you next see HITS-SW?

We developed and tested HITS-SW as a modification to the HITS ranking algorithm. Assigning similarity weights to all the links in a collection of web pages allowed us to run a connectivity-analysis that considered the content of the documents. By analyzing the content in a set of documents, we hoped to achieve a better analysis of the connectivity; specifically, we hoped to mitigate the following problems existing in HITS:

- HITS often could not distinguish between citation links and navigation links between documents, and thus assumed each type was equally important.
- HITS allowed citation structure to generalize a specific search and return results that were not similar to the original query.

Results showed trends suggesting that using similarity in addition to link structure produces regular patterns that are distinct from HITS. The resulting rankings are not necessarily universally better for the average user, but these interesting modifications might improve the final result:

- We could change how we chose the parameter for similarity ranges. When selecting the range of acceptable similarity weight for the parameter for HITS-SW, we would choose a value based upon the aggregate data. Distribution of weights would often have various plateaus where many documents would have the exact same similarity. We hypothesize that often those documents that receive the same similarity weight are in the same cluster. We found a number of examples where a cluster of documents were judged to be similar, and all received the same weighting. One example of this was PowerPoint slideshows that appeared to be similar because they all contained certain words: “next”, “previous”, “slide”, etc. Although it would require additional computation, we believe that using similarity weights based upon the weights of the relevant pages might produce different, possibly interesting, results.
- We could remove similar results by using clustering algorithms. Using a standard method for clustering documents, we would minimize the number of duplicate documents and hopefully those with similar locality. This could be achieved by simplifying the topology of the Web by collapsing multiple documents within a given domain, server, or directory before using HITS-SW for ranking. This might ensure greater differences between HITS-SW and HITS and perhaps the patterns in HITS-SW would become clarified.
- We could change the method for collecting keywords from documents. It might be wise to consider different portions of a document as different levels of importance. The text in the title might be much more important than other content in the document. Content in the middle of the document might be more important than navigational buttons at the top or left side of the screen. While it might be difficult to extract actual content from navigational and other structural content, this would be an area worth investigating.

With these modifications, we think that we could more precisely determine whether the patterns in our data consistently produce results that are different from HITS. These

should also allow a better determination whether the results from HITS have legitimately improved, as the themes in the rankings should be more pronounced.

Our hope is that future studies will be able to use this information to produce a refined algorithm that is more careful in obtaining precise similarity data to produce targeted results. The preliminary results, however, show that including similarity information in a connectivity-based algorithm produces greatly modified results.

## References

- [1] <http://www.loria.fr/~bonhomme/sw/stopword.en> (current 7 Apr. 2001)
- [2] “A Standard for Robot Exclusion,”  
<http://info.webcrawler.com/mak/projects/robots/robots.html> (current 7 Apr. 2001)
- [3] “Acquiring Perl Software,” <http://www.perl.com/pub/language/info/software.html>  
(current 7 Apr. 2001)
- [4] “Lynx Information,” <http://lynx.browser.org> (current 7 Apr. 2001)
- [5] “Oxford English Dictionary Online,” <http://dictionary.oed.com/entrance.dtl> (current 7 Apr. 2001)
- [6] “Search Engine Sizes,” <http://searchenginewatch.internet.com/reports/sizes.html>  
(current 7 Apr. 2001)
- [7] J.M. Baccash, *Linkage Spam Filtering for the Hub Authority Algorithm*, junior paper, Dept. Computer Science, Princeton University, Princeton, N.J., 1999.
- [8] K. Bharat, M.R. Henzinger, “Improved Algorithms for Topic Distillation in a Hyperlinked Environment,” *Proc. 21st International ACM SIGIR Conference on Research and Development in Information Retrieval*, Aug. 1998, pp.104-111.
- [9] S. Brin, L. Page, “The PageRank Citation Ranking: Bringing Order to the Web,” unpublished manuscript, Dept. Computer Science, Stanford University, Stanford, Calif., 1998.
- [10] S. Brin, L. Page, “The Anatomy of a Large-Scale Hypertextual Web Search Engine,” <http://www-db.stanford.edu/~backrub/google.html> (current 7 Apr. 2001)
- [11] R.A. Brualdi, *Introductory Combinatorics*, ed. 3, Prentice-Hall, Upper Saddle River, N.J., 1999.
- [12] S. Chakrabarti, et al., “Automatic resource compilation by analyzing hyperlink structure and associated text,” *Proc. 7th International World Wide Web Conference*, Brisbane, Australia , 1998, pp. 65-74.
- [13] S. Chakrabarti, et al., “Mining the Web’s Link Structure,” *IEEE Computer*, vol. 32, no. 8, Aug. 1999, pp. 60-67.
- [14] J. Cho, et al., “Efficient crawling through URL ordering,” *Proc. of 7th World Wide Web Conference*, 1998.
- [15] T. Christiansen, N. Torkington, *Perl Cookbook*, O’Reilly & Associates, Sebastopol, CA, 1998.
- [16] J. Downes, “Pepper and Salt,” *The Wall Street Journal*, 4 Dec. 2000.
- [17] G.W. Gurnas, et al., “The Vocabulary Problem in Human-System Communication,” *Communications of the ACM*, vol. 30, no. 11, Nov 1987, pp. 964-971]

- [18] D. Hawking, et al., “Results and Challenges in Web Search Evaluation,” *Proc. of 8th World Wide Web Conference*, 1999.
- [19] M.R. Henzinger, et al., “Measuring Index Quality Using Random Walks on the Web,” *Proc. of the 8th International World Wide Web Conference*, Toronto, May 1999, pp. 213-225.
- [20] J.M. Kleinberg, “Authoritative sources in a hyperlinked environment,” *Proc. 9th Annual ACM-SIAM Symposium on Discrete Algorithms.*, 1998.
- [21] R.R. Korfhage, *Information Storage and Retrieval*, John Wiley & Sons, New York, 1997.
- [22] C.L. Giles, S. Lawrence, “Accessibility of Information on the Web,” *Nature*, 8 Jul. 1999, pp. 107-109.
- [23] P. Pirolli, et al., “Silk from a Sow’s Ear: Extracting Usable Structures from the Web,” *Proc. of ACM SIGCHI ’96*, pp. 118-125.
- [24] M.F. Porter, “An algorithm for suffix stripping,” *Program*, 1980, pp. 14:130-137.

## A Aggregate Entropy Data

### Or, Will the reader examine indistinguishable numbers?

Definitions for these terms may be found in section 4.3.2. Each set of seven rows describes the aggregate entropy data for a specific query: the first four correspond to HITS-SW *hi* and *lo* queries using similarity weights `mean_12` and `mean_u`; the latter three rows refer to data from the mean, median, and standard deviation for the random trials.

- Column One denotes whether the query corresponds to the Computer Science or the Princeton domain.
- Column Two corresponds to actual query. Note that “‘a’ ‘b’” means ‘a’ AND ‘b’ whereas “‘a b’” means the phrase ‘a b’.
- Columns Three and Four describe the type of ranking for the query. Note that the median values may come from different random runs as—e.g., the median value for the inversions may come from random trial 3 while the median value for the distance may come from random trial 17.
- Columns Five and Six refer to the number of elements ranked. “Total” refers to the number of elements in the HITS query, and “new” to the row.
- Columns Seven and Eight refer to the number of inversions found in the top ten (“t10”) elements and the original top ten elements (“orig”).
- Columns Nine and Ten refer to the amount of intersection with the original HITS query in the top ten (“t10”) and top twenty (“t20”) ranked elements.
- Column Eleven denotes the distance the top ten elements traveled.

Table 1: Aggregate Entropy

domain, query string, query type				elements		inversions		intersection		dist
				total	new	t10	orig	t10	t20	t10
cs	‘felten’	hi	m12	215	158	4	1	3	13	15.25
cs	‘felten’	hi	mu	215	141	7	2	3	12	16.72
cs	‘felten’	lo	m12	215	208	21	4	5	14	12.86
cs	‘felten’	lo	mu	215	192	10	3	3	9	16.55
cs	‘felten’	rand	mean	215	194.88	6.92	4.6	8.24	17.76	5.65
cs	‘felten’	rand	med	215	195	7	5	8	18	5.66
cs	‘felten’	rand	stddev	0	3.61	1.8	1.78	0.72	1.01	1.12
cs	‘graph’	hi	m12	408	275	14	3	5	5	19.26
cs	‘graph’	hi	mu	408	271	9	3	5	6	18.03
cs	‘graph’	lo	m12	408	358	9	6	8	13	8.2
cs	‘graph’	lo	mu	408	348	10	7	8	12	8.04
cs	‘graph’	rand	mean	408	350.12	2.6	1.76	8.8	14.08	3.83
cs	‘graph’	rand	med	408	349	2	2	9	14	3.79
cs	‘graph’	rand	stddev	0	4.87	1.73	1.36	0.65	1.47	2.03
cs	‘graphics’	hi	m12	3912	3623	14	0	0	2	32.41
cs	‘graphics’	hi	mu	3912	3488	11	9	7	16	12.22
cs	‘graphics’	lo	m12	3912	3537	2	0	4	12	10.06
cs	‘graphics’	lo	mu	3912	3678	1	1	4	11	11.56
cs	‘graphics’	rand	mean	3912	3585.76	11.36	3.8	6.6	16.08	8.96
cs	‘graphics’	rand	med	3912	3586	10	3	7	16	9.38
cs	‘graphics’	rand	stddev	0	19.85	4.49	2.33	0.87	1.68	1.3

Table 1: Aggregate Entropy *continued*

domain, query string, query type				elements		inversions		intersection		dist
				total	new	t10	orig	t10	t20	t10
cs	'herbach'	hi	m12	4	4	0	0	3	3	0
cs	'herbach'	hi	mu	4	3	0	0	2	2	1
cs	'herbach'	lo	m12	4	4	2	2	3	3	1.79
cs	'herbach'	lo	mu	4	4	1	1	3	3	1
cs	'herbach'	rand	mean	4	4	0.72	0.72	3	3	0.7
cs	'herbach'	rand	med	4	4	1	1	3	3	1
cs	'herbach'	rand	stddev	0	0	0.68	0.68	0	0	0.63
cs	'java'	hi	m12	2734	2406	0	0	0	0	50.79
cs	'java'	hi	mu	2734	2188	6	2	6	7	16.93
cs	'java'	lo	m12	2734	2395	4	2	7	13	6.99
cs	'java'	lo	mu	2734	2379	7	2	5	9	11.04
cs	'java'	rand	mean	2734	2512.08	3.76	2.52	8.04	11.24	6.5
cs	'java'	rand	med	2734	2511	4	3	8	12	6.35
cs	'java'	rand	stddev	0	10.54	2.15	1.58	0.68	1.54	1.66
cs	'language'	hi	m12	1092	770	10	0	0	8	33.99
cs	'language'	hi	mu	1092	738	4	0	3	9	16.05
cs	'language'	lo	m12	1092	973	5	3	6	11	10.78
cs	'language'	lo	mu	1092	842	12	6	5	8	14
cs	'language'	rand	mean	1092	962.04	12.2	2.28	5.52	15.32	10.86
cs	'language'	rand	med	1092	964	12	2	6	15	10.65
cs	'language'	rand	stddev	0	8.2	4.1	2.01	1.08	1.31	1.53
cs	'language' 'java'	hi	m12	526	406	23	3	3	16	16.25
cs	'language' 'java'	hi	mu	526	381	4	0	3	9	16.05
cs	'language' 'java'	lo	m12	526	448	5	3	6	11	10.78
cs	'language' 'java'	lo	mu	526	389	12	6	5	8	14
cs	'language' 'java'	rand	mean	526	476.88	12.16	2.24	5.52	15.32	10.84
cs	'language' 'java'	rand	med	526	479	12	2	6	15	10.65
cs	'language' 'java'	rand	stddev	0	5.19	4.04	1.94	1.08	1.31	1.52
cs	'lapaugh'	hi	m12	112	73	2	2	7	12	12.24
cs	'lapaugh'	hi	mu	112	84	4	4	7	10	12.79
cs	'lapaugh'	lo	m12	112	112	13	4	5	10	14
cs	'lapaugh'	lo	mu	112	110	10	2	4	8	13.16
cs	'lapaugh'	rand	mean	112	100.52	5.96	5.52	9	14.68	4.94
cs	'lapaugh'	rand	med	112	101	6	6	9	15	4.88
cs	'lapaugh'	rand	stddev	0	3.11	2.52	2.24	0.65	1.99	1.43
cs	'microsoft windows'	hi	m12	22	13	13	2	5	11	11.67
cs	'microsoft windows'	hi	mu	22	11	16	1	4	10	12.51
cs	'microsoft windows'	lo	m12	22	19	9	5	7	17	10.07
cs	'microsoft windows'	lo	mu	22	22	21	17	9	19	10.42
cs	'microsoft windows'	rand	mean	22	18.48	11	5.72	7.16	16.64	7.94
cs	'microsoft windows'	rand	med	22	19	11	5	7	17	7.88
cs	'microsoft windows'	rand	stddev	0	1.42	4.56	3.31	0.8	1.47	1.1
cs	'network'	hi	m12	788	597	6	0	0	0	29.41
cs	'network'	hi	mu	788	575	6	1	7	11	9.46
cs	'network'	lo	m12	788	699	12	11	8	16	9.65
cs	'network'	lo	mu	788	606	3	3	7	17	9.79
cs	'network'	rand	mean	788	707.6	12.28	10.16	8.84	15.6	7.31
cs	'network'	rand	med	788	708	12	10	9	16	7.44
cs	'network'	rand	stddev	0	8.6	5.14	4.17	0.69	1.29	1.47
cs	'problem'	hi	m12	1354	829	23	0	2	7	32.03
cs	'problem'	hi	mu	1354	822	21	2	5	5	24.64
cs	'problem'	lo	m12	1354	1226	13	9	6	13	11.64
cs	'problem'	lo	mu	1354	1101	10	3	5	11	12.66
cs	'problem'	rand	mean	1354	1134.56	10.44	8.64	8.8	14.8	7.01
cs	'problem'	rand	med	1354	1131	10	8	9	15	6.95
cs	'problem'	rand	stddev	0	11.79	4.46	4.01	0.41	1.12	1.52
cs	'security'	hi	m12	608	503	13	0	0	0	36.51
cs	'security'	hi	mu	608	464	9	0	4	18	13.3
cs	'security'	lo	m12	608	563	2	1	3	5	15.64

Table 1: Aggregate Entropy *continued*

domain, query string, query type				elements		inversions		intersection		dist
				total	new	t10	orig	t10	t20	t10
cs	'security'	lo	mu	608	504	5	1	3	4	15.63
cs	'security'	rand	mean	608	564.36	10.52	2.6	5.52	13.84	10.45
cs	'security'	rand	med	608	565	10	2	6	14	10.58
cs	'security'	rand	stddev	0	7.38	2.97	1.89	1.23	1.72	1.8
cs	'security' 'java'	hi	m12	361	310	3	0	0	5	35.5
cs	'security' 'java'	hi	mu	361	295	7	0	3	18	14.42
cs	'security' 'java'	lo	m12	361	333	2	1	3	5	15.64
cs	'security' 'java'	lo	mu	361	318	5	1	3	4	15.63
cs	'security' 'java'	rand	mean	361	338.88	10.72	2.68	5.56	13.84	10.39
cs	'security' 'java'	rand	med	361	339	10	2	6	14	10.54
cs	'security' 'java'	rand	stddev	0	5.04	2.82	1.91	1.26	1.72	1.81
cs	'shrimp'	hi	m12	24	19	3	1	8	17	6.7
cs	'shrimp'	hi	mu	24	18	15	10	8	15	7.98
cs	'shrimp'	lo	m12	24	12	12	2	3	8	16.01
cs	'shrimp'	lo	mu	24	17	11	4	4	13	13.61
cs	'shrimp'	rand	mean	24	21.04	13.36	6.28	6.56	16.92	9.87
cs	'shrimp'	rand	med	24	21	13	5	6	17	10.07
cs	'shrimp'	rand	stddev	0	1.14	6.93	5.18	1.23	0.57	1.97
cs	'solaris'	hi	m12	102	80	2	0	0	4	20.93
cs	'solaris'	hi	mu	102	65	18	0	1	2	27
cs	'solaris'	lo	m12	102	80	9	0	5	9	8.78
cs	'solaris'	lo	mu	102	81	4	4	8	16	5.91
cs	'solaris'	rand	mean	102	89.6	14.52	0.88	2.56	8.36	16.19
cs	'solaris'	rand	med	102	89	15	0	0	8	20.44
cs	'solaris'	rand	stddev	0	2.99	6.62	1.59	3.15	4.1	6.25
cs	'theory'	hi	m12	359	206	12	1	4	6	18.1
cs	'theory'	hi	mu	359	186	7	1	2	3	24
cs	'theory'	lo	m12	359	347	8	3	6	15	9.28
cs	'theory'	lo	mu	359	317	8	4	7	15	8.22
cs	'theory'	rand	mean	359	308.64	4.52	3.24	7.56	14.24	6.32
cs	'theory'	rand	med	359	311	4	3	7	14	6.33
cs	'theory'	rand	stddev	0	8.15	2.2	1.98	0.87	1.42	1.75
pr	'admissions'	hi	m12	2165	787	21	10	7	18	10.34
pr	'admissions'	hi	mu	2165	1132	13	4	7	13	8.12
pr	'admissions'	lo	m12	2165	2154	12	1	7	15	6.91
pr	'admissions'	lo	mu	2165	2050	11	1	7	15	6.75
pr	'admissions'	rand	mean	2165	1893.92	10.48	5.84	8	16.52	6.4
pr	'admissions'	rand	med	2165	1895	11	6	8	17	6.67
pr	'admissions'	rand	stddev	0	11.35	3.85	3.57	0.65	0.87	1.36
pr	'basketball'	hi	m12	1254	430	9	0	1	1	25.58
pr	'basketball'	hi	mu	1254	423	1	0	1	1	22.55
pr	'basketball'	lo	m12	1254	1252	11	4	7	12	9.96
pr	'basketball'	lo	mu	1254	1140	6	4	5	13	10.84
pr	'basketball'	rand	mean	1254	1059.92	13.08	2.64	5	11.88	12.55
pr	'basketball'	rand	med	1254	1061	12	2	5	11	12.85
pr	'basketball'	rand	stddev	0	11.43	4.4	2.63	1.44	1.62	2.17
pr	'chapel'	hi	m12	1282	191	31	3	4	7	22.09
pr	'chapel'	hi	mu	1282	628	4	0	0	0	40.97
pr	'chapel'	lo	m12	1282	1279	0	0	0	0	31.78
pr	'chapel'	lo	mu	1282	1173	17	0	0	0	37.88
pr	'chapel'	rand	mean	1282	1077.4	13.52	2	3.48	7.16	20.15
pr	'chapel'	rand	med	1282	1079	13	1	4	10	12.57
pr	'chapel'	rand	stddev	0	13.78	6.33	2.92	3.07	6.07	12.07
pr	'cit'	hi	m12	1776	1045	37	0	0	4	26.97
pr	'cit'	hi	mu	1776	509	14	2	3	8	19.33
pr	'cit'	lo	m12	1776	1416	0	0	2	2	26.52
pr	'cit'	lo	mu	1776	1760	13	2	4	4	23.3
pr	'cit'	rand	mean	1776	1369.08	16.84	0.68	2.2	5.08	23
pr	'cit'	rand	med	1776	1372	17	0	2	5	21.87

Table 1: Aggregate Entropy *continued*

domain, query string, query type				elements		inversions		intersection		dist
				total	new	t10	orig	t10	t20	t10
pr	'cit'	rand	stddev	0	15.11	5.18	1.11	1.32	2.5	4.59
pr	'class'	hi	m12	14568	6484	19	2	3	6	16.77
pr	'class'	hi	mu	14568	6565	2	0	3	7	13.42
pr	'class'	lo	m12	14568	12687	10	1	7	12	11.05
pr	'class'	lo	mu	14568	13463	14	2	7	11	11.52
pr	'class'	rand	mean	14568	12356.36	8.68	2.8	5.96	10.8	10.99
pr	'class'	rand	med	14568	12363	9	3	6	11	11.03
pr	'class'	rand	stddev	0	37.26	3.56	1.58	0.89	1.61	1.45
pr	'dinky'	hi	m12	216	56	27	6	5	17	15.08
pr	'dinky'	hi	mu	216	152	14	5	7	20	7.7
pr	'dinky'	lo	m12	216	215	18	3	6	20	10.67
pr	'dinky'	lo	mu	216	108	8	0	0	0	32.89
pr	'dinky'	rand	mean	216	197.6	17.88	6	6.32	19.08	10.16
pr	'dinky'	rand	med	216	198	19	6	6	19	10.07
pr	'dinky'	rand	stddev	0	2.97	4.75	3.63	1.14	0.49	1.25
pr	'dormnet'	hi	m12	338	91	5	0	5	14	9.53
pr	'dormnet'	hi	mu	338	92	5	0	3	12	12.04
pr	'dormnet'	lo	m12	338	336	6	4	6	9	10.29
pr	'dormnet'	lo	mu	338	320	7	4	8	13	6.88
pr	'dormnet'	rand	mean	338	294.44	6.8	4.04	6.92	14.96	7.42
pr	'dormnet'	rand	med	338	294	7	3	7	15	7.49
pr	'dormnet'	rand	stddev	0	4.27	3.3	2.7	1	1.86	1.05
pr	'felten'	hi	m12	81	10	10	0	2	5	17.01
pr	'felten'	hi	mu	81	24	16	2	6	10	9.68
pr	'felten'	lo	m12	81	81	19	9	8	16	9.99
pr	'felten'	lo	mu	81	79	17	12	8	10	11.39
pr	'felten'	rand	mean	81	60.04	16.64	10.28	7.36	11.2	10.48
pr	'felten'	rand	med	81	60	19	11	8	12	11.33
pr	'felten'	rand	stddev	0	3.09	8.59	7.57	1.89	2.31	3.91
pr	'forbes'	hi	m12	924	227	9	0	0	2	33.83
pr	'forbes'	hi	mu	924	394	8	5	9	12	4.33
pr	'forbes'	lo	m12	924	920	6	3	8	18	5.08
pr	'forbes'	lo	mu	924	867	17	0	0	0	25.17
pr	'forbes'	rand	mean	924	779.72	12.24	3.72	5.84	11.48	11.29
pr	'forbes'	rand	med	924	782	9	4	8	15	7.38
pr	'forbes'	rand	stddev	0	9.99	6.77	2.94	3.54	6.96	8.14
pr	'frist'	hi	m12	680	260	3	0	2	7	17.4
pr	'frist'	hi	mu	680	254	11	0	1	1	34.2
pr	'frist'	lo	m12	680	676	12	0	2	7	28.56
pr	'frist'	lo	mu	680	634	6	1	8	15	6.29
pr	'frist'	rand	mean	680	569.72	8.04	3.16	6.56	15.48	8.17
pr	'frist'	rand	med	680	571	8	3	7	16	7.65
pr	'frist'	rand	stddev	0	7.15	3.55	2.79	1.26	1.23	1.65
pr	'glee'	hi	m12	215	55	13	0	2	3	26.03
pr	'glee'	hi	mu	215	93	20	0	1	3	28.97
pr	'glee'	lo	m12	215	216	28	0	0	4	33.11
pr	'glee'	lo	mu	215	177	4	0	0	0	35.13
pr	'glee'	rand	mean	215	183.52	21.36	1.88	5.64	11.6	15.44
pr	'glee'	rand	med	215	183	22	2	5	11	16.07
pr	'glee'	rand	stddev	0	4.26	5.28	1.3	1.15	1.89	4.36
pr	'graph'	hi	m12	716	182	26	0	0	0	42.3
pr	'graph'	hi	mu	716	304	3	0	0	0	42.48
pr	'graph'	lo	m12	716	714	0	0	10	20	0
pr	'graph'	lo	mu	716	668	14	0	0	0	31.57
pr	'graph'	rand	mean	716	629.84	16.72	2.36	3.2	8.56	19.27
pr	'graph'	rand	med	716	629	18	1	3	9	17.93
pr	'graph'	rand	stddev	0	6.88	6.83	2.48	1.71	3.55	5.03
pr	'graphics'	hi	m12	7792	4742	1	0	2	3	33.46
pr	'graphics'	hi	mu	7792	1619	1	0	2	5	32.49

Table 1: Aggregate Entropy *continued*

domain, query string, query type				elements		inversions		intersection		dist
				total	new	t10	orig	t10	t20	t10
pr	'graphics'	lo	m12	7792	7754	6	0	0	0	45.64
pr	'graphics'	lo	mu	7792	7347	5	0	0	0	45.63
pr	'graphics'	rand	mean	7792	6960.04	4.56	2.64	7.8	15.2	5.38
pr	'graphics'	rand	med	7792	6956	5	2	8	15	5.38
pr	'graphics'	rand	stddev	0	27.01	2.18	1.82	0.87	1.26	1.29
pr	'herbach'	hi	m12	35	9	19	8	5	8	11.18
pr	'herbach'	hi	mu	35	10	16	6	6	9	9.43
pr	'herbach'	lo	m12	35	34	0	0	9	19	0.79
pr	'herbach'	lo	mu	35	27	7	7	8	14	7.45
pr	'herbach'	rand	mean	35	28.52	4.96	2.6	8.08	14.92	4.63
pr	'herbach'	rand	med	35	28	5	2	8	15	4.38
pr	'herbach'	rand	stddev	0	1.94	2.13	1.55	0.57	1.15	1.44
pr	'java'	hi	m12	1254	533	27	0	0	0	47.81
pr	'java'	hi	mu	1254	709	8	0	0	0	46.43
pr	'java'	lo	m12	1254	1203	1	1	10	20	1
pr	'java'	lo	mu	1254	1171	10	0	0	0	38.18
pr	'java'	rand	mean	1254	1120.72	17.44	1.6	2.64	6.92	20.77
pr	'java'	rand	med	1254	1121	17	1	3	7	19.19
pr	'java'	rand	stddev	0	9.21	8.37	1.96	1.44	3.16	4.84
pr	'language'	hi	m12	7456	2723	6	0	0	0	42.52
pr	'language'	hi	mu	7456	3241	6	0	0	0	51.53
pr	'language'	lo	m12	7456	7236	0	0	10	20	0
pr	'language'	lo	mu	7456	6959	3	0	0	0	46.98
pr	'language'	rand	mean	7456	6589.84	18.12	0.4	1.28	3.36	28.29
pr	'language'	rand	med	7456	6590	19	0	1	4	25.96
pr	'language'	rand	stddev	0	20.44	6.55	0.91	1.14	1.91	7.77
pr	'language' 'java'	hi	m12	238	18	23	0	0	1	33.51
pr	'language' 'java'	hi	mu	238	101	13	0	0	6	32.75
pr	'language' 'java'	lo	m12	238	237	0	0	8	18	7.92
pr	'language' 'java'	lo	mu	238	209	13	0	2	2	26.4
pr	'language' 'java'	rand	mean	238	197.96	18.16	1.52	2.64	9.6	19.1
pr	'language' 'java'	rand	med	238	197	18	1	3	10	17.5
pr	'language' 'java'	rand	stddev	0	5.01	6.72	1.61	1.5	2.38	5.55
pr	'lapaugh'	hi	m12	31	10	12	4	6	7	9.64
pr	'lapaugh'	hi	mu	31	18	14	1	6	17	8.78
pr	'lapaugh'	lo	m12	31	31	16	4	5	17	12.52
pr	'lapaugh'	lo	mu	31	30	24	2	3	11	17.88
pr	'lapaugh'	rand	mean	31	28.96	15.8	3	5.64	16.72	10.67
pr	'lapaugh'	rand	med	31	29	17	3	5	17	11.91
pr	'lapaugh'	rand	stddev	0	1.17	7.45	1.71	0.99	1.28	2.19
pr	'menu'	hi	m12	5630	1936	28	0	0	0	55.23
pr	'menu'	hi	mu	5630	1194	15	0	0	0	37.78
pr	'menu'	lo	m12	5630	5598	0	0	0	0	51.95
pr	'menu'	lo	mu	5630	5374	3	0	0	0	51.95
pr	'menu'	rand	mean	5630	5011.24	8.64	2.84	6.36	13	8.48
pr	'menu'	rand	med	5630	5010	8	2	6	13	8.38
pr	'menu'	rand	stddev	0	20.07	3.23	2.27	1.11	1.87	1.83
pr	'microsoft windows'	hi	m12	282	88	13	0	0	3	34.87
pr	'microsoft windows'	hi	mu	282	88	3	0	0	0	34.36
pr	'microsoft windows'	lo	m12	282	279	11	2	8	13	6.71
pr	'microsoft windows'	lo	mu	282	268	2	2	10	19	2
pr	'microsoft windows'	rand	mean	282	258.4	9.24	0.4	2.8	5.16	18.95
pr	'microsoft windows'	rand	med	282	258	9	0	3	5	19.13
pr	'microsoft windows'	rand	stddev	0	4.09	4.8	0.76	1.19	1.82	2.78
pr	'nassoons'	hi	m12	110	19	20	5	6	10	14.18
pr	'nassoons'	hi	mu	110	59	15	0	2	4	20.35
pr	'nassoons'	lo	m12	110	110	11	2	6	13	11.54
pr	'nassoons'	lo	mu	110	93	14	7	7	14	10.84
pr	'nassoons'	rand	mean	110	96	9.36	4.32	5.92	14.2	10.49

Table 1: Aggregate Entropy *continued*

domain, query string, query type				elements		inversions		intersection		dist
				total	new	t10	orig	t10	t20	t10
pr	'nassoons'	rand	med	110	96	10	4	6	14	10.62
pr	'nassoons'	rand	stddev	0	2.48	4.35	2.87	1.26	1.58	2.35
pr	'network'	hi	m12	6917	2175	19	0	0	0	45.96
pr	'network'	hi	mu	6917	3093	9	5	9	10	6.17
pr	'network'	lo	m12	6917	6850	4	4	9	20	4.58
pr	'network'	lo	mu	6917	6512	12	0	0	0	32.19
pr	'network'	rand	mean	6917	6229.2	13.08	0.04	0.24	0.6	28.64
pr	'network'	rand	med	6917	6225	12	0	0	0	29.38
pr	'network'	rand	stddev	0	21.59	4.65	0.2	1.2	3	4.82
pr	'president'	hi	m12	4908	1168	11	1	3	3	18.63
pr	'president'	hi	mu	4908	1736	2	1	4	5	13.54
pr	'president'	lo	m12	4908	4877	8	2	5	11	14.94
pr	'president'	lo	mu	4908	4475	13	0	5	13	16.68
pr	'president'	rand	mean	4908	3944.08	10.48	2.52	5.2	10.88	12.67
pr	'president'	rand	med	4908	3942	11	3	5	11	12.68
pr	'president'	rand	stddev	0	22.74	3.55	1.5	1.04	2.33	2.05
pr	'problem'	hi	m12	10204	2361	15	2	3	14	15.27
pr	'problem'	hi	mu	10204	4178	3	2	7	18	8.04
pr	'problem'	lo	m12	10204	10112	5	1	3	5	17.64
pr	'problem'	lo	mu	10204	9471	24	0	0	0	48.43
pr	'problem'	rand	mean	10204	8161.8	8.52	1.4	4.56	8.12	12.44
pr	'problem'	rand	med	10204	8774	9	1	4	8	13.06
pr	'problem'	rand	stddev	0	2147.91	3.4	1.47	1.45	2.54	3.23
pr	'prospect'	hi	m12	3113	506	2	0	5	6	15.74
pr	'prospect'	hi	mu	3113	1436	7	0	1	1	33.69
pr	'prospect'	lo	m12	3113	3106	3	0	0	0	36.91
pr	'prospect'	lo	mu	3113	2906	23	0	0	0	51.93
pr	'prospect'	rand	mean	3113	2607.68	8.44	1.64	4.92	9.2	24.3
pr	'prospect'	rand	med	3113	2613	6	1	8	15	5.5
pr	'prospect'	rand	stddev	0	17.11	6.01	1.8	4.2	7.94	23.1
pr	'security'	hi	m12	4985	1398	7	0	0	0	45.58
pr	'security'	hi	mu	4985	2316	1	1	5	6	21.49
pr	'security'	lo	m12	4985	4957	7	1	6	19	7.26
pr	'security'	lo	mu	4985	4662	23	0	0	3	22.28
pr	'security'	rand	mean	4985	4137.72	14.08	0.08	0.24	2.16	24.11
pr	'security'	rand	med	4985	4305	15	0	0	2	25.49
pr	'security'	rand	stddev	0	862.28	4.92	0.4	1.2	1.62	6.06
pr	'security' 'java'	hi	m12	218	29	16	0	0	0	32.44
pr	'security' 'java'	hi	mu	218	111	33	0	0	8	33.14
pr	'security' 'java'	lo	m12	218	217	2	2	9	20	2.5
pr	'security' 'java'	lo	mu	218	198	4	0	0	0	28.64
pr	'security' 'java'	rand	mean	218	190.4	17.76	1.48	2.64	9.32	18.92
pr	'security' 'java'	rand	med	218	190	19	1	3	10	17.13
pr	'security' 'java'	rand	stddev	0	3.74	6.21	1.64	1.47	3.52	4.65
pr	'sex'	hi	m12	1967	194	21	0	0	0	49.27
pr	'sex'	hi	mu	1967	1182	3	0	0	0	47.88
pr	'sex'	lo	m12	1967	1948	1	1	10	20	1
pr	'sex'	lo	mu	1967	1843	8	0	1	1	35.99
pr	'sex'	rand	mean	1967	1704.68	18.52	1.04	2.28	5.44	24.35
pr	'sex'	rand	med	1967	1707	18	0	2	5	25.47
pr	'sex'	rand	stddev	0	11.13	4.99	1.77	1.49	1.56	4.07
pr	'shrimp'	hi	m12	75	14	10	2	5	6	16.46
pr	'shrimp'	hi	mu	75	44	3	0	2	7	20.78
pr	'shrimp'	lo	m12	75	75	0	0	0	7	19.04
pr	'shrimp'	lo	mu	75	69	3	0	5	10	13.5
pr	'shrimp'	rand	mean	75	62.32	9.84	1.44	3	9.32	15.66
pr	'shrimp'	rand	med	75	63	8	1	3	9	16.31
pr	'shrimp'	rand	stddev	0	3.44	5.81	1.66	1.83	1.46	3.43
pr	'solaris'	hi	m12	349	84	4	0	1	1	25

Table 1: Aggregate Entropy *continued*

domain, query string, query type				elements		inversions		intersection		dist
				total	new	t10	orig	t10	t20	t10
pr	'solaris'	hi	mu	349	126	18	0	0	0	25.99
pr	'solaris'	lo	m12	349	343	15	4	5	17	13.79
pr	'solaris'	lo	mu	349	333	2	1	9	19	3.79
pr	'solaris'	rand	mean	349	311.44	20.32	6.2	5.76	14.36	12.29
pr	'solaris'	rand	med	349	311	21	6	5	14	13.37
pr	'solaris'	rand	stddev	0	4.34	7.44	3.39	1.01	1.44	2.59
pr	'sprint' 'football'	hi	m12	81	45	7	3	4	4	17.27
pr	'sprint' 'football'	hi	mu	81	26	9	1	2	6	21.39
pr	'sprint' 'football'	lo	m12	81	81	26	0	1	9	21.41
pr	'sprint' 'football'	lo	mu	81	74	8	5	8	16	5.67
pr	'sprint' 'football'	rand	mean	81	70.64	14.12	1.44	4.84	13.2	11.99
pr	'sprint' 'football'	rand	med	81	71	14	1	5	13	12.19
pr	'sprint' 'football'	rand	stddev	0	2.4	6.33	1.12	1.11	1.53	2
pr	'theory'	hi	m12	4084	711	32	0	0	0	49.01
pr	'theory'	hi	mu	4084	1766	17	0	0	0	47.18
pr	'theory'	lo	m12	4084	4057	3	3	10	18	2.79
pr	'theory'	lo	mu	4084	3759	26	0	0	8	21.48
pr	'theory'	rand	mean	4084	3404.6	22.2	0.2	0.84	4.32	24.87
pr	'theory'	rand	med	4084	3408	22	0	1	4	25.04
pr	'theory'	rand	stddev	0	17.75	4.39	0.5	0.85	1.35	2.3
pr	'visual'	hi	m12	2117	422	23	0	0	0	37.57
pr	'visual'	hi	mu	2117	597	24	0	0	0	39.57
pr	'visual'	lo	m12	2117	2102	3	3	9	20	3.38
pr	'visual'	lo	mu	2117	1947	9	7	9	20	6.25
pr	'visual'	rand	mean	2117	1726.92	11.08	6.28	7.36	16.96	8.14
pr	'visual'	rand	med	2117	1727	11	6	7	17	8.04
pr	'visual'	rand	stddev	0	16.56	3.48	3.35	0.91	1.24	1.62
pr	'whales'	hi	m12	223	12	9	0	2	2	29.25
pr	'whales'	hi	mu	223	56	9	2	4	4	23.27
pr	'whales'	lo	m12	223	223	25	0	2	2	28.6
pr	'whales'	lo	mu	223	215	15	3	3	3	26.28
pr	'whales'	rand	mean	223	141.16	23.16	0.44	1.76	3.44	27.69
pr	'whales'	rand	med	223	141	23	0	1	3	27.77
pr	'whales'	rand	stddev	0	6.82	6.11	0.96	1.16	1.26	2.42