CATSSEARCH: A NEW SEARCH ENGINE DESIGN
FOR WEB PAGES IN THE UCCS DOMAIN

by

JING YANG

A thesis submitted to the Graduate Faculty of the
University of Colorado at Colorado Springs
in partial fulfillment of the
requirements for the degree of
Masters of Science
Computer Science
2005
This thesis for Master of Science degree by

Jing Yang

has been approved for the
department of Computer Science

by

______________________
Jugal K. Kalita, Chair

______________________
Maria F. Augusteijn

______________________
Xiaobo Zhou

Date
Yang, Jing (M.S., Computer Science)

CatsSearch: A New Search Engine Design for Web Pages in the UCCS Domain

Thesis directed by Professor Jugal K. Kalita

The present paper outlines the development and testing of CatsSearch, a search engine that is designed to crawl and index web pages within the www.uccs.edu domain, and to provide more efficient and accurate search results. CatsSearch adopts Google’s concept of link popularity and PageRank for ordering result pages for a given search query. CatsSearch also utilizes information for keyword location, keyword proximity, and keyword frequency to provide accurate search results. Finally, the search engine is compared against Yahoo and Google within the UCCS domain for 25 test items. Results suggest that CatsSearch, within the very limited conditions of the present study, is faster than Google and Yahoo in terms of retrieval, and is roughly as accurate. However, CatsSearch only returned results on web pages, not on other file types.
CONTENTS

CHAPTER

1. INTRODUCTION .................................................................................................................1
  1.1 The UCCS Search Engine ............................................................................................1
  1.2 Background Research .................................................................................................2
    1.2.1 About Google .........................................................................................................4
    1.2.2 About Yahoo .........................................................................................................5
    1.2.3 Issues in large scale crawling ...............................................................................6
    1.2.4 Issues in indexing ....................................................................................................7
    1.2.5 Issues in ranking result pages ...............................................................................8
    1.2.6 Issues in improving search efficiency .................................................................9
    1.2.7 Issues in user search experience .........................................................................10
    1.2.8 Issues in search engines comparison ....................................................................11

2. DESIGN GOAL ...................................................................................................................13
  2.1 Extracting Keywords and Corresponding Information .............................................13
  2.2 Utilizing Anchortext .....................................................................................................13
  2.3 Ranking Pages ...............................................................................................................14

3. IMPLEMENTATION .........................................................................................................16
  3.1 System Architecture ....................................................................................................16
3.2 Algorithms

3.2.1 Crawling

3.2.2 Obtaining distinct words for each page

3.2.3 Extracting links

3.2.4 Storing anchor text for inbound links

3.2.5 Ranking pages

3.2.6 Searching pages

3.3 Database

3.4 Problems Faced and Solutions

4. EVALUATING THE EFFICIENCY AND EFFECTIVENESS OF CATSSEARCH

4.1 Methodology

4.1.1 Evaluating search response times and number of “hits” per query

4.1.2 Evaluating the relevance and accuracy of search results

4.2 Results

4.2.1 Search times and number of “hits” per query

4.2.2 Relevance and accuracy of search results

4.3 Discussion

5. CONCLUSION

5.1 Problems and Limitations with CatsSearch

5.2 Suggested Future Work to Improve CatsSearch

5.3. Thoughts about Evaluating Accuracy and Relevance
TABLES

Table

1. Weighted Points Assignment ................................................................. 22
2. URLs ....................................................................................................... 33
3. EXTRACTEDWORDS .............................................................................. 34
4. ANCHORWORDS .................................................................................. 34
5. BASEURLS ............................................................................................ 35
6. DISPLAYWDS ...................................................................................... 35
7. STOPWORDS .......................................................................................... 36
8. Average search time and number of “hits” for each item searched .......... 49
FIGURES

Figure

1. CatsSearch interface window ................................................................. 15
2. Schematic representation of the high level CatsSearch architecture .......... 18
3. CatsSearch ER Diagram ........................................................................... 32
4. Screen capture of CatsSearch .................................................................. 40
5. Average search times (in seconds) ............................................................. 47
6. Average number of hits ............................................................................. 48
7. Average ranking ......................................................................................... 51
CHAPTER 1
INTRODUCTION

1.1 The UCCS Search Engine

As the present project developed, the UCCS search engine used the ht://Dig system, an open source searching system for a domain, or intranet. This type of system was developed at the San Diego State University. ht://Dig uses both "exact" and "fuzzy" matching. The rank of a match is determined by the weight of the words that caused the match and the weight of the algorithm that generated the match (ht://Dig group, 1995-2004; Salton & McGill, 1983; Yale University Library…, 1996-98). Although ht://Dig had some of its own features such as the fuzzy algorithms (including soundex algorithm, metaphone algorithm, etc.), users often found it quite disappointing when doing a search using the ht://Dig engine. For example, when one typed the keywords “search engine” to look for web pages regarding how a search engine works, one received a list of pages with the first page (highest rank) titled “XML - What’s in it for us.” It fully described XML but one could not find what one was really looking for in the page. Such shortcomings inspired the present project, and may have led to the recent adoption by UCCS of a Google-powered search engine. Even though the present project developed as an attempt to improve keyword retrieval in relation to the original ht://Dig system, but now will be compared to the current Google-powered system, and to Yahoo within the UCCS domain.
The present section will briefly review issues that are critical to search engine development, beginning with a general overview, and then addressing more specific issues related to search engine efficiency and accuracy.

1.2 Background Research

Search engines use automated software programs known as spiders to survey the Web. Web pages are retrieved and analyzed by these programs. Data collected from each web page are then added to a search engine index. The Web is growing so fast and web pages are updated so frequently that it forces the search engine to revisit web pages frequently. To scale with the ever-growing Web, many search engines use distributed web crawling to improve their system effectiveness. The effectiveness of an information system is the ability to furnish information services that the users need.

The distributed system operates by having individual computers located in multiple locations crawl web pages at the same time, or by having computers connected to the Internet crawl internet address in the background. These crawled web pages are downloaded and compressed, and sent back to central servers, which manage a large database. Such distributed systems may even involve an ad-hoc form of collaboration by enlisting volunteers to join the effort using their home computers. The quality of returned pages is another factor that affects a search engine’s system effectiveness. Every search engine employs unique algorithms to rank web pages. The algorithms weigh various factors, such as a page's design and links. Page-related factors include page format, use of tags, and so on. Outside factors include outgoing links and inbound links. By
constantly refining and improving their algorithms, search engines hope to give their users the most relevant results.

There are two kinds of system tests: *effectiveness* and *efficiency*. The effectiveness of an information system is measured by its ability to furnish information services that the users need. The efficiency, on the other hand, is a measure of the cost or the time necessary to perform a given set of tasks (Salton & McGill, 1983). The design goal of CatsSearch is to provide good system effectiveness and efficiency.

There are two traditional measurements for effectiveness of information retrieval: recall and precision. Recall measures the proportion of relevant information actually retrieved in response to a search (i.e., the number of relevant items actually obtained divided by the total number of relevant items contained in the collection). Precision measures the proportion of retrieved items actually relevant (i.e., the number of relevant items actually obtained divided by the total number of retrieved items; Salton & McGill, 1983). Recall and precision have trade off: indexing with high recall tends to retrieve most potentially relevant items; at the same time, however, precision may suffer because some marginally relevant items are also likely to be retrieved when many different subject terms are covered by the index terms. Since the number of documents on the web has been increasing dramatically, but people’s ability to view the documents has not, people still tend to look at the first ten returned items (i.e., the first page of search results). Because of this, obtaining high precision pages becomes more important and desirable. The goal of the current search engine is to obtain such a high precision by making use of link structures on web pages. Web structure has been increasingly used in information retrieval, and many search engines have used hyperlinks to improve web
page ranking. The reasoning here is based on the assumption that links and the text describing them usually have a close relationship with the documents they are on and that they links link to.

1.2.1 About Google. When Google’s search engine rose to prominence around 2001, its successful ranking algorithm promptly drew public attention and has since spawned a number of imitators. PageRank is a numeric value that represents how important a page is on the web. Google figures that when one page links to another page, it is essentially casting a vote for the other page. The more votes that are cast for a page, the more important the page must be. Also, the importance of the page that is casting the vote determines how important the vote itself is. Google calculates a page's importance from the votes cast for it. The importance of each vote is taken into account when a page's PageRank is calculated. In other words, how many other web sites and web pages link to a given page is taken into consideration with PageRank. The PageRank of linking pages and the number of links on these pages contribute to the PageRank of the linked page. This makes it possible for Google to order its results according to each page’s popularity. From the above definition, we can see that if page A has many citations and/or those citation pages have high PageRank then A, in turn, will have very high PageRank.

Many search engines associate the text of a link with the page that the link is on. In addition, Google associates it with the page the link points to. The personal nature of the anchortext allows for connecting words to destination pages (inbound anchortext). Inbound Anchortext has two advantages. First, it provides more accurate descriptions of
linked web pages than do the pages themselves. Second, it allows non-text web pages such as programs, images, and so on, to be returned--these cannot be crawled by a text based search engine.

The Google search engine provides high quality pages to the users. In addition, Google provides many of services, such as news and image searches (and many others). For the present study, the most important service is the ‘Google University Search,’ which offers free SiteSearch and WebSearch services to any nonprofit organization, including accredited universities or educational institutions, which apply. WebSearch searches the internet, while SiteSearch allows one to search within a specific domain requested by the applier. Google allows the search box to reside on the local web site. But the search result pages are hosted by Google with a customized look created by the applier at sign up. Pages within the specified domain are updated with Google's main crawl about once a month (https://services.google.com/pss_faq.html#6). According to one source, Google has indexed 114,000 pages within the UCCS domain (The Googlizer, 2005).

1.2.2 About Yahoo. The Yahoo search engine originally used the Inktomi indexing service. Inktomi relies on title, keywords (meta tag) and text to sort search results. The Inktomi Spider Slurp gives highest priority for keywords in the page title. It also gives a higher priority for keywords in META tags. It severely penalizes consecutive repetition of keywords in META tags, and keywords that do not appear on the page. Slurp will re-index a site about once every three weeks. According to one source, Yahoo has indexed 24,500 pages within the UCCS domain (The Googlizer, 2005).
In 2002, Yahoo dropped Inktomi for Google because of its extra range and better ranking of web pages. It combined its own original human-compiled directory listings with Google crawler-based search results. This was a significant change. Yahoo used to provide the sites it lists with a human-reviewed title and description. Google, in contrast, will automatically generate one based on the content of the web page. Since this is not a perfect system, Yahoo's listings may be more readable whereas Google’s listings provide better ranking results. Ultimately, a combination of both is better.

1.2.3 Issues in large scale crawling. Given the modern information explosion, it is not surprising that the World Wide Web has also been growing at a tremendous rate. In a recent study, Lyman and Varian (2003), estimated the total size of the “surface web” (i.e., those web pages that are publicly available) to be about 167 Terabytes (TB); by extension, the “deep web” (i.e., dynamic, database driven websites) is estimated to be as large as 91,850 TB. In terms of search engines, it has been noted that, as of January 2003, approximately 319 million searches are conducted daily with the major search engines (SearchEngineWatch.com). Obviously, with this tremendous growth, the challenge facing search engines is considerable, particularly with regards to their ability to find, download and index the most appropriate web pages (Najork & Wiener, 2001).

Several strategies are available for getting the crawler to crawl only the best pages. Cho, Garcia-Molin, and Page (1998), for example, have performed a series of tests to crawl pages in the stanford.edu domain (about 179,000 pages). Several ordering techniques were used, including breadth-first (i.e., pages are crawled in the order they are
discovered), back-link count (i.e., pages with highest number of links are crawled first), PageRank (i.e., pages with the highest page rank are crawled first), and random (i.e., uncrawled pages are randomly chosen) methods. Their general conclusion was that the PageRank method seemed to work the best, followed by the breadth-first method.

More recently, Najork and Wiener (2001) noted that, even though the PageRank method may direct the crawler to high-quality pages, computing PageRank values for millions of web pages is extremely expensive. By comparison, the breadth-first method is much more economical because it does not need to retrieve the full-page content; rather, it only needs to extract the links from each page. They tested their breadth-first search crawling metric on a much larger scale (328 million pages) than Cho et al. (1998), and found that the breadth-first method not only retrieved high-quality pages first, but also that the quality of pages decreased as the crawling proceeded. Najork and Wiener observed that, even though PageRank provided somewhat higher-quality pages to the crawler, the breadth-first method provided a reasonable bias towards high quality pages in a more efficient manner.

1.2.4 Issues in indexing. One of the major problems facing search engines is to answer several thousand queries per second within a reasonable response time (Long & Suel, 2005). To meet the challenge of this tremendous workload, efficient indexing is crucial. One of the most common means of improving index scalability is caching, an efficient and beneficial technique (Long & Suel). Caching may be manifested as a two-level system or a three-level system. The two-level caching technique collects the results of
repeated queries at the front end while index data for frequently queried terms are cached at a lower level by keeping such terms in memory. By caching recently returned query results, one can remove repeated queries from the search engine’s workload and increase the amount of information the search engine can process (Long & Suel). Thus, two-level caching can ultimately increase overall system performance, both in terms of speed and in terms of throughput (Saraiva, de Moura, Ziviani, Meira, Fonseca, & Ribeiro-Neto, 2001).

Three-level caching is like two-level catching, but it adds an additional, intermediate level of caching that caches stored pairs of terms that commonly occur together in queries. This type of three-level caching is knows as intersection or projection caching (depending on the implementation), and has been tested recently in a large web crawl (Long & Suel, 2005). This architecture utilizes a main characteristic of all major search engines, namely that they return pages by default with all search words in them. Taking advantage of this, the three-level architecture caches the pairwise intersection between pages, which significantly improves system performance, provided that one has good caching policy selection criteria.

1.2.5 Issues in ranking result pages. The link structure of web pages is very important for search engines, especially in organizing results pages. One of the most efficient algorithms for ranking results is PageRank, which was developed by Google. However, a critical issue with the PageRank algorithm is that it ignores dangling pages during page rank calculations; it removes the dangling pages until the last few iterations of PageRank
are calculated (Eiron, McCurley, & Tomlin, 2004). Unfortunately, these dangling pages are far from trivial, as they tend to be rather pervasive and often turn out to be very high quality pages. To cite one example, most PostScript and PDF files, although they may have high quality content, do not contain embedded outlinks; they thus form a relatively large proportion of dangling pages. Eiron et al. (2004) refine the PageRank algorithm by taking into account the dangling links during the whole period of PageRank calculation. In crawling over 1 billion pages, they allowed dangling pages to jump to a randomly selected page from every dangling page. After the test, they found that the existence of dangling pages could have significant effects on the ranking of non-dangling pages.

1.2.6 Issues in improving search efficiency. Search engines process user queries usually using two type of indices: the frequency index keeps track of how often a particular term appears on a document, and the positional index keeps track of where the term appears in the document. As almost all of the cost of processing queries is due to accessing these indices, the key to speed up overall system performance is to decrease the storage cost as well as the time to access to this storage. This is usually done by compress the data in the indices. One such compression method is the pruning method, which is based on lossy compression, by which the potentially useless items (such as stop words) are removed from the index without negatively affecting the overall quality of the result pages.
As noted by de Moura et al. (2005), Carmel et al. (2001) proposed a pruning method in which each single term is submitted to the search engine as a query. The resulting pages are those related to the term and they are sorted according their importance by the search engine. The top portion of these result pages guides the pruning: “each frequency index entry that represents the occurrence of term \( t \) in document \( D \) is removed from the index if \( D \) is not present in the top portion” (de Moura et al., 2005, p. 237) the result pages.

But Carmel’s method does not consider the occurrence of terms across documents, which may cause information loss when conjunctive or phrase terms occur in both the pages present in the top portion and the bottom portion of the result pages. Based on this observation, de Moura et al. refined Carmel’s method by obtaining the significant sentences of document \( D \) such that each sentence contains at least one of the significant terms of \( D \) (terms occur in \( D \), and \( D \) appears in the top portion of result pages as defined in Carmel’s test). These significant sentences guide the pruning process: “In the positional index, only occurrences of terms in the selected sentences are preserved. The remaining are removed. In the frequency index, entries that represent the frequency of a term \( t \) in a document \( D \) are preserved only if \( t \) occurs at least once in at least one of the selected sentences of \( D \)” (de Moura et al., 2005, p. 238). In testing this technique, de Moura et al. noted a significant improvement by achieving a compression rate of 60% while still maintaining a similar ranking to that computed with the original indices.
1.2.7 Issues in user search experience. One topic that affects all search engines is the search engine interface. Much of a search engine’s effectiveness, in fact, depends on this interface with the user, more and more of whom are very inexperienced. Until recently, little research has addressed this aspect of search engine performance (Topi & Lucas, 2003). Most users today probably know very little about Boolean logic, and rely almost exclusively on simple search boxes (Pollock & Hockley, 1997; Turtle, 1994) even though more advanced options are readily available. A potential solution proposed by Topi and Lucas is an assisted interface for users, and/or appropriate training, assisted users in forming correct Boolean queries, thereby obtaining more accurate search results. Search engine development in the future will no doubt benefit from better-designed user interfaces.

1.2.8 Issues in search engines comparison. The existing measurements for comparing search engines performance include the most commonly used and most traditional measurements: Precision and Recall (as mentioned above in Section 1.2). However, recall is hard to achieve since “it requires the knowledge of the total number of relevant pages in the collection,”(Vaughan, 2004, p. 679). Precision is always used in formal information retrieval experiments. Web page precision measurement can be done in different ways: it can be done by using binary relevance judgments, by which a page is decided to be either relevant or not relevant; or it can be done by using more discrete judgments, with which web pages are judged according to multi-level gradations of judgment. Such discrete ranking systems can range from 3-levels (Shang & Li, 2002), as
used in the present study, to 4- or 5-levels (Gwizdka & Chignell, 1999; Su, Chen, &
dong, 1998).

Vaughan (2004) takes this one step further by proposing a continuous ranking
system, whereby web pages are rank ordered from most relevant to least relevant. With
this measurement, the correlation between human ranking and search engine ranking is
calculated. The higher the correlation coefficient, the closer the search engine ranking is
to the human ranking, and thus the better the search engine performance. This proposal
based on the assumption that “human subjects are able to make relevance judgments on a
continuous scale” (Vaughan, 2004, p. 678), and on the assumption that the web pages
returned are typically ranked.
CHAPTER 2
DESIGN GOAL

2.1 Extracting Keywords and Corresponding Information

CatsSearch provides an alternative searching system for the UCCS domain. It utilizes full text crawling of web pages to help build a database index; it also extracts each word from a web page along with information about the word’s location, field, font, etc. Words are assigned different points according to their location and/or appearance. For example, words that appear in a title obtain higher points than words in a paragraph. These combined total points contribute to the ranking of the page they are on.

2.2 Utilizing Anchortext

CatsSearch makes heavy use of link structure, since some pages’ content does not provide obvious clues. For example, the home page of the Microsoft Corporation (http://www.microsoft.com) provides no mention of the fact that they sell operating systems (Glover, Tsioutsioulakis, Lawrence, Pennock, & Flake, 2002). Anchor text, on the other hand, is decided by people who are interested in the page, and thus is usually better at summarizing the content of the page. This has been shown in some studies. For example Blum (as cited in Glover et al., 2002) compared two systems for computer
science web pages, one for full-text, and one for the words on the links pointing in to the target pages. These results indicate that anchor text alone is slightly less powerful than the full-text alone, and that the combination is better. Google is another example that utilizes inbound anchor text; Google allows pages to be returned based on keywords occurring in inbound anchor text, even if the words don’t occur on the page itself, such as returning http://www.yahoo.com/ for a query of “web directory” (Glover et al., 2002).

Based on this study, CatsSearch’s crawler extracts anchor text, and assigns the text high points. These points contribute to the pages the links point to as well as to the pages they are on. In addition to utilizing anchor text, CatsSearch also extracts extended anchor text (words that appear near anchor text) and nearby headings. This is done because these words are likely related to the anchor text, and thus also provide descriptive sentences about the target pages.

2.3 Ranking Pages

CatsSearch adopts Google’s concept and implements the PageRank algorithm to bring result pages in a more efficient order. According to the algorithm, a page is ranked according to the quality and quantity of sites that point to it. In theory, the more sites that link to yours, the higher your ranking in the search engine results will be because the greater number of links indicates a higher level of popularity among users of the Internet. To implement this algorithm, we need to know the number of links that each page has and all the pages that point to each page. We can obtain each page’s links during crawling, and store these data in a table in the database. After crawling is finished, we can obtain each page’s back links by querying the table.
The CatsSearch interface window is depicted in Figure 1. The interface also has a help center link to provide search assistance.

Figure 1. CatsSearch interface window.
CHAPTER 3
IMPLEMENTATION

3.1 System Architecture

This section provides a high level overview of how the system works, as schematically depicted in Figure 2. The following sections will discuss the software algorithm and data structure that are not mentioned in this section. Most of CatsSearch is implemented in perl.

In CatsSearch, the web crawling is done by the Crawler. The Crawler starts at a given seed url (http://www.uccs.edu) and does several things. First, it fetches the web page from the server, it extracts desired text words and stores them as IndexedWords in the database along with each word’s location, field, url, and which page the word is on. This information will be used later to calculate the word’s weight, which will contribute to the page’s rank rate. Secondly, the Crawler obtains all the hyperlinks on the page. The extracted links are added to the list to be visited. The program then visits each url recursively. Each examined url is stored in the database as IndexedUrls. IndexedUrls is for indexing urls and it does not store duplicate urls. Third, in addition to extracting and storing the anchor text in the database as the IndexedWords, the crawler extracts all anchor text and its corresponding target url and stores them as the InboundWords in the database.
DistWdIndexer gets unique words from IndexedWords for each page, and stores them as DistinctWords in the database table. The purpose of recording these words is to speed up search process and thus reduce search response time. As an alternative, the Crawler could extract these distinct words during crawling, as shown in Figure 2.

LinkExtractor essentially works the same way as the Crawler. It is different, however, in that it does not examine pages. It only collects information about all the links on each page it visits, and this information is later used to calculate each page’s page rank.

The InboundIndexer will later add the InboundWords to the IndexedWords but with their target url as the url, and the url that they are on as the baseurl. This actually adds more points to the target urls. Putting all data together in this manner will speed up the searching process.

After all the links are extracted, Pageranker uses these data to calculate each page’s pagerank, and then puts this information in the IndexedUrls according to each page’s corresponding url. The explanation of how pagerank is calculated is provided below.

Finally, Search uses the IndexedWords and IndexUrls to carry out search queries.
Figure 2. Schematic representation of the high level CatsSearch architecture. The dotted arrow indicates that the Crawler can also extract DistinctWords during crawling.
3.2 Algorithms

There are five software modules that are used by CatsSearch: crawl.pl, insertinbounds.pl, prlinks.pl, pr.pl, and search.php. This section provides an algorithm for each module as well as explanations and examples as necessary.

Web crawler is an essential component to search engines. Search engines use the crawler to traverse the Internet and to parse each of the individual pages passed along the way. Because of the large quantity of pages on the Web, it is essential to crawl the Web in an efficient manner, maintaining a reasonable measure of quality and “freshness” This is referred to as the crawling policy. CatsSearch’s crawling policy includes the following, which applies to all algorithms provided in this document.

1. It examines web pages that are within the www.uccs.edu domain.
2. Keywords are extracted from the pages with a .(s)htm(l) extension if they do have extension.
3. URLs that include #, ?, %, -, ’ are ignored.
4. Web pages that are not returned by the servers within 30 seconds are considered stale pages and are not revisited again.
5. Some pages returned by servers do not have actual content; they are removed from the index and are not fetched again.
6. Links on a page that point to the page itself and/or are the same as other hyperlinks are ignored.
7. All text words are converted into lower case before they are stored in the database.
8. Stop words are indexed but are used only for phrase queries.

3.2.1 Crawling. The crawler starts a loop at a given url (http://www.uccs.edu/). It first checks to see if it is a crawled url. If it has not been crawled, it obtains the content of the page using the LWP::UserAgent module; then it crawls the hyperlinks on the page using the HTML::LinkEtor module. For each link, the crawler checks its absoluteness and file type. It also checks to see if it is the same page as the url or is a duplicate of another url in the links. This is mainly for crawling efficiency and for page rank calculation, which will be explained in the following paragraphs. Then, the crawler calls upon the subroutine extractwords(), which examines the url and extracts all text words from it. After examining the page, the crawler recursively visits each link on the page if the page is in the UCCS domain.

CRAWL (url, baseurl)

input: raw web page p to be crawled, list of crawled pages C(url, baseurl)
output: insert list of indexed words into table extractedwords, list of indexed urls stored in database table urls, list of anchor words stored in database table anchorwords

length of C <- n
crawled <- F
for k <- 1 to n
  do if C[k]{url} = url
      then crawled <- T
      break
f <- fetch content of url
if f is successful
  then newLinks[] <- obtain links of url
  foreach newurl in newLinks[]
    do absolutize newurl
      remove if is not required file type
      remove if newurl=ur
      remove if is a duplicate in newLinks
    do if crawled =F
      then extractwords (f, url, baseurl)
    else store in urls with note='timeout'
do if url in uccs domain
    then for q <- 1 to length of newLinks
        crawl (newLinks[q], url)

extractwords(f, url, baseURL)

insert into urls
obtains tags of the url
foreach tag of tags
    do if tag = 'a'
        then store anchor text into extractedwords
        get inbound link of the anchortext
        store anchor text into anchorwords
        store extended anchor text into anchorwords
        else store text words into extractedwords

return

The subroutine extractedwords is the core that does most of the work of crawling. It first stores the url in table urls, along with its information (e.g., number of links, file size, file type, update time). It then obtains all the tags on the page using the HTML::TokeParser module. For each tag, it extracts the text words and stores them in the table extractedwords. This table also contains each word’s field name, points assigned to it, and the urlid from which page the word was extracted. If the tag is an anchor tag, in addition to storing anchor text into extractedwords, the crawler obtains the inbound links pointed to by the anchor text, and stores the anchor text and its extended anchortext (the text that occurs near the heading) and its inbound url into the table anchorwords. These words, including their target urlid, field and assigned points, are eventually put into extractedwords after crawling is finished, with the value of urlid of its target url as the urlid value. Text that occurs near and nearby headings (extented anchor text) is extracted and stored into the database in the same way.
The assignment of weight points to each word is not shown in the above algorithm because words in different fields obtain different points. Basically, words are assigned points between 20 and 0.1. Words that receive the maximum points are those in the field url, title, meta, h1, h2, and so on. Minimum points are assigned to words that are not in significant fields such as tag p. Numbers that are in between 0.1 and 20 are assigned to words accordingly. This system is illustrated in Table 1.

**Table 1. Weighted Points Assignment**

<table>
<thead>
<tr>
<th>points</th>
<th>field names</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>url, title, meta, h1, h2, span-head, span-spotlight-head, inbound-anchor</td>
</tr>
<tr>
<td>10</td>
<td>anchor,</td>
</tr>
<tr>
<td>5</td>
<td>h3, span-style24,</td>
</tr>
<tr>
<td>2</td>
<td>extend-inbound, extend-inbound-head, h4, h5</td>
</tr>
<tr>
<td>0.2</td>
<td>b, strong, em, big</td>
</tr>
<tr>
<td>0.1</td>
<td>everything else</td>
</tr>
</tbody>
</table>

The strategy of this weighting method is to make a larger gap between number 20 and 0.1, allowing us to always get pages with search words in url, title, and so on. Whenever we can obtain these pages, other pages are basically ignored. If no words are found in these significant fields, the smaller numbers come into play. Notice each field name is kept separately in the database even though it has the same weighting points. By doing so, we can easily update the database when we need to adjust these field points to obtain better search results.
The CRAWL algorithm first goes through a large loop in which it parses each current url, gets the content of the url, and then extracts the index words. If we assume the total number of pages is n, and each url contains m links and t tags, then it takes time $tn$ to execute the first loop. Because the recursive call in each loop takes time $mn$, the total time that this algorithm takes is $cn(m+t)$, where c is a constant.

3.2.2 Obtaining distinct words for each page. GETDISTWDS picks each unique word for each page from the table `extractedwords` and store them in table `distwds`. The algorithm is shown below:

```plaintext
GETDISTWDS
input database table extractedwords
output database distwds

urlids[] <- select distinct urlid from extractedwords
foreach urlid in urlids[]
    words[] <- select distinct words from extractedwords where urlid = urlid
    store words[] in distwds where urlid = urlid
```

This algorithm goes through a simple loop, so the execution time is $cn$ where c is a constant and n is the number of pages obtained from `extractedwords`.

3.2.3 Extracting links. The algorithm EXLINKS is designed to extract web page links. These links are later used by the algorithm PRCalculation to calculate each page’s page rank. EXLINKS is similar to CRAWL, but it does not extract index words from the content of the page, nor does it ignore duplicate web pages and crawled pages. The reason is that, in order to get backlinks for calculating page rank for a url, we need to get all the baseurls that have links pointing to this url, which may require visiting the same
url more than one time. The same reason applies for visiting crawled urls, as long as
the pair of baseurl and url do not have duplicates. That is why the table baseurls has a
composite primary key.

**EXLINKS (url, baseurl)**

input: url, baseurl
list of crawled urls C selected from database table baseurls (urlid, url, numoflinks, baseurl)

output: list of urls inserted into database table baseurls(urlid, url, numoflinks, baseurl)

length of C <- n

crawled <- F

for k <- 1 to n
    do if url = C[k]{url} and baseurl = C[k]{baseurl}
        then crawled <- T
        break

do if crawled = F
    do
        f <- fetch content of url
        do if fetch successful
        then N <- number of links extracted from url
        insert into table baseurls(urlid, url, N, baseurl)
        do if url is in uccs domain
        then newList [ ] <- links of url
        for q <- 1 to length of newList[
            do absolutize newList[q]
            exLinks (newList[q], url)

The algorithm first checks if the pair of url and baseurls is crawled; if not, it
fetches the content of the url, and obtains the number of links of the url using
HTML::LinkExtor. It then stores the names of the url, baseurl, and the number of links in
the url into the table baseurls. It then obtains all the links of the url if it is in the UCCS
domain. Since this algorithm is only different than crawling in that it does not extract
keywords from each page, it takes the time cmn where n is total number of pages, m is
number of links each page contains, and c is a constant.
3.2.4 Storing anchor text for inbound links. During crawling, the anchor text of web pages is extracted and stored in `extractedwords`; in addition, the anchor text is also stored in the table `anchorwords` for the sake of the target web pages. The reason for not storing anchor text directly into `extractedwords` is that we do not know the target web pages’ `urlids`, which we can obtain by querying table `urls` after the crawling is finished. Putting all indexed words together makes searching more efficient. This algorithm is designed for this purpose.

**insertinbounds()**

input: database tabel anchorwords(urlid, anchorword, anchorurl)

            database table urls (urlid, url, …)

output: updated anchorwords
        new urls inserted into table urls

urlID <- select max(urlid) from urls
select anchorword, anchorurl from table anchorwords
for each anchorword, anchorurl
    do select urlid from table urls where url=anchorurl
    do if urlid exists
        then update anchorwords set anchuid=urlid
        else @head <- get header of the anchorurl
            do if @ head exists
                then urlID++
                insert into urls(urlID, anchorurl, @head)
                else go to next loop

The algorithm first selects each anchorword and its corresponding anchorurl from the table anchorwords. Then, from table urls, it obtains the urlid for this anchorurl; if the urlid exists, it updates anchorwords to assign the obtained urlid to the anchoruid. If the urlid does not exist, it means this anchorurl was not crawled by the crawler. To be able to return this page in response to a user query, this anchorurl needs to be added to `urls` with
its corresponding newly assigned urlID. This algorithm goes through a simple loop; it takes linear time cn, where n is the number of rows returned from the table anchorwords.

### 3.2.5 Ranking pages

Since CatsSearch is designed to use Google’s PageRank algorithm to rank its result pages, it is important to examine Google’s PageRank algorithm (Brin & Page, no date):

We assume page A has pages T1...Tn which point to it (i.e., are citations). The parameter is a damping factor which can be set between 0 and 1. We usually set d to 0.85. There are more details about d in the next section. Also C(A) is defined as the number of links going out of page A. The PageRank of a page A is given as follows:

\[
PR(A) = (1-d) + d \left( \frac{PR(T1)}{C(T1)} + \ldots + \frac{PR(Tn)}{C(Tn)} \right)
\]

As we can see from the above algorithm, the page’s rank is calculated by adding up each of its inbound link’s page rank, divided by this links’ number of outgoing links. The data stored in table `baseurls` provide the links information needed to carry out the calculation.

The details of the algorithm are explained below:

```
Pagerank()
input: database table urls(urlid, url, ...)
        database table baseurls (urlid, url, numoflinks, baseurl)
output: calculated page rank for each url and inserted into table urls according to each corresponding urlid
array of hashes <- L
j <- 0
select url from urls
for each url selected
   do L[j]{url} <- url
      L[j]{pr} <- 0
      L[j]{numofinbounds} <- 0
      select baseurl from baseurls where baseurls.url = L[j]{url}
      k <- 0
      for each baseurl selected
```
To be specific, the Pagerank procedure works as follows: It first obtains urls from the table URLS; to each url obtained, it assign an initial page rank value 0. It then queries the table baseurls to get all the pages (baseurls/backlinks) that have links pointing to this url. It queries, for each backlink, the table baseurls again to get the number of links that exist in this backlink. After looping all the backlinks, it assigns the total number of backlinks as the numofinbounds of the current url.

After looping all pages obtained from urls, it goes through another 50 cycles of loops. In each loop, it calls a subroutine calpr(), which does the page rank calculations. The first loop takes kn time, where n is the number of pages to be calculated, and k is number of inbound links that each page have. The second loop calls the subroutine
calpr() 50 times for each page. Because the subroutines calpr() take the same time as the first loop \( kn \), the second loop costs \( kn^2 \). The total cost of this algorithm is \( kn^2 \).

In Google’s PageRank algorithm, each page’s rank is calculated according to the page rank of each inbound link. In the first loop, all url page ranks are assigned to 0. We can use this numbers first. When it runs the loop a couple of times, this initial number changes because we use a damping factor of 0.85 for each calculation; this changed value will again be used for the next calculation.

Below is a simple example of how the algorithm works:

suppose we have two pages P1 and P2, P1 has a link to P2 and P2 has a link to P1. So we have 
\[ C(P1) = 1 \quad \text{and} \quad C(P2) = 1 \]
first we assign the initial page rank for them:
\[ \text{pr}(P1) = 0 \]
\[ \text{pr}(P2) = 0 \]
first calculation:
\[ \text{PR}(P1) = 0.15 + 0.85 * 0 \]
\[ = 0.15 \]
\[ \text{PR}(P2) = 0.15 + 0.85 * 0.15 \]
\[ = 0.2775 \]
and we use these two numbers for the next calculation:
second calculation:
\[ \text{PR}(P1) = 0.15 + 0.85 * 0.2775 \]
\[ = 0.385875 \]
\[ \text{PR}(P2) = 0.15 + 0.85 * 0.385875 \]
\[ = 0.47795375 \]
and again:
third calculation:
\[ \text{PR}(P1) = 0.15 + 0.85 * 0.47795375 \]
\[ = 0.5562946875 \]
PR(P2) = 0.15 + 0.85 * 0.5562946875
= 0.622850484375

In this manner, both numbers will keep going up. This indicates that no matter where we start our guess, once the calculation settles down, the ‘normalized probability of the distribution (the average page ranks for all pages) will be one.

3.2.6 Searching pages. This algorithm takes user input search words, parses them and queries database tables *urls* and *extractedwords* to obtain web pages that contain all the words in them. It returns to users the result pages according to their ranking rate. If it is a phrase search, it further queries extractedwords to get each word’s location to make sure all search words contained in each page are next to each other and in the correct order. The algorithm is shown below:

```sql
search(words[])

input: search word(s) from user query, table *urls*, table *extractedwords*
output: web pages returned to user

create temporary table *tmp1*
create temporary table *tmp2*

select into *tmp1* urlid, sum(points) from *extractedwords* that contains words[0]

for j <- 1 to number of words to be searched
    *tmp2* <- select urlid, sum(points) from *extractedwords* that contains words[j] and urlid in *tmp1*
    truncate tmp1
    *tmp1* <- *tmp2*
    truncate tmp2

*urls[]* <- select url from *urls*, *tmp1* where urlid = *tmp1*.urlid order by *tmp1*.points
return *urls[]*

//if is phrase search
select into *tmp1* distinct urlid from *extractedwords* where word=words[0]

for j <- 1 to number of words to be searched
    *tmp2* <- select distinct urlid from *extractedwords* that contains words[j] and urlid in *tmp1*
truncate tmp1
tmp1 <- tmp2

truncate tmp2
for each urlid in tmp1
  totalpoints <- 0
  select loc, points <- location from extractedwords with words[0] and urlid
  for each loc returned
    location <- loc+1;
    for j <- 1 to number of search words
      select loc <- location from extractedword with the required url, word, location
      do if select result is 0
        then p <- F
        break
      else location <- loc+1
    do if p != F
      then totalpoints <- totalpoints + points * number of words
  do if (totalpoints > 0) {
    then urlids[] <- urlid, points
  urls[] <- select url from urls, urlids where urlid = urlids.urlid order by urlids.points
  return urls[]

Let T be the time each time to query database table, then the total cost of regular search is cnT where c is constant, n is the number of search words. Suppose the number of result pages is m, then for phrase search, it takes nm^2 to further check a word’s location for each word in each page. So the total cost of a phrase search is cnT plus cnm^2, which is cn(T+m^2).

3.3 Database

A total of six tables are used by CatsSearch to store data in the database, as shown in Figure 3, which shows the entities and relationships (ER) for CatsSearch. Two of them are maintained by the crawler. URLs stores data at the url level. It includes each page’s urlid, url, type, size, pagerank, lastupdated, lastcrawled and note. Pagerank will be calculated once crawling is finished. Note is used to track when a web page cannot be returned because of a timeout, or when the page returned does not actually have content. In that case, the same page will not be fetched again. This is mainly for crawling
efficiency. The table EXTRACTEDWORDS stores data at the individual word’s level. It includes each word’s wordid, word, urlid, location, field, and points. The foreign key urlid in extractedwords connects each word to its web page in urls. Attribute location is used when a user issues a phrase query. Each word’s field not only provides how many points each word gains but is also used by the advanced search when field information is required. These two tables are the ones that are used to carry out search queries.

The other three tables are BASEURLS, ANCHORWORDS, and STOPWORDS. BASEURLS has the attributes baseurl, url, numoflinks and urlid. This table is used to calculate each page’s pagerank. The explanation of how pagerank is calculated is provided below. BASEURLS has a composite primary key (baseurl, url) because, in order to obtain inbound links for a url page, we want to keep all different baseurls that link to this url, but exclude duplicate baseurl-url pairs. Once each page’s pagerank is calculated, it is put in table URLS at its corresponding urlid.

Table ANCHORWORDS has the attributes urlid, anchorwd, anchorurlid, anchorurl, and urlid. This table is used to catch the anchorwords for the target urls. After crawling, these data are added to table EXTRACTEDWORDS but with anchorurlid as the urlid, and urlid as the baseurlid. This actually adds more points to the target urls. In addition, putting all data in the same table will increase the speed of the searching process.
Figure 3. CatsSearch ER Diagram
Table DISPLAYWDS stores titlewds and displaywds for each urlid. These words are displayed as a short summary of the page when result pages are returned to users.

The last table is a look-up table, STOPWORDS, which stores 30 common stop words. These words are ignored when the crawler calculates each word’s weight and when inboundIndexer adds InboundWords to the IndexedWords since they are not counted when returning results to user’s queries.

The table structures of these five tables are shown below:

**Table 2. URLs**

(ulid int primary key, url varchar(5000), numoflinks int, size int, type varchar(500),
lastupdated varchar(1000), lastcrawled varchar(1000), pagerank real, note varchar(50))

<table>
<thead>
<tr>
<th>ulid</th>
<th>url</th>
<th>numoflinks</th>
<th>size</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td><a href="http://www.uccs.edu/">http://www.uccs.edu/</a></td>
<td>17</td>
<td>-1</td>
<td>text/html</td>
</tr>
<tr>
<td>1</td>
<td><a href="http://www.uccs.edu/campusinfo/campusinfo_form.htm">http://www.uccs.edu/campusinfo/campusinfo_form.htm</a></td>
<td>3</td>
<td>5330</td>
<td>text/html</td>
</tr>
<tr>
<td>43875</td>
<td><a href="http://www.uccs.edu/proto2005/alumni_sub.shtml">http://www.uccs.edu/proto2005/alumni_sub.shtml</a></td>
<td>37</td>
<td>-1</td>
<td>text/html</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>lastupdated (varchar)</th>
<th>lastcrawled (varchar)</th>
<th>pagerank (real)</th>
<th>note (varchar)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wed Dec 31 17:00:00 1969</td>
<td>Fri Oct 28 03:48:53 2005</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>Thu Aug 11 09:24:42 2005</td>
<td>Fri Oct 28 03:48:42 2005</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>Wed Dec 31 17:00:00 1969</td>
<td>Sat Oct 29 20:12:56 2005</td>
<td>0</td>
<td>-</td>
</tr>
</tbody>
</table>
Table 3. EXTRACTEDWORDS
(wordid bigint primary key, word varchar(1000), field varchar(100), points int,
urlid bigint)

<table>
<thead>
<tr>
<th>wordid</th>
<th>word</th>
<th>field</th>
<th>points</th>
<th>urlid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>uccs</td>
<td>url</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>5599</td>
<td>colorado</td>
<td>b</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>8888307</td>
<td>navigation</td>
<td>anchor</td>
<td>100</td>
<td>44414</td>
</tr>
</tbody>
</table>

foreign key urlid references URLs(urlid) on update cascade and on delete cascade

Table 4. ANCHORWORDS
(wordid bigint primary key, anchorwd varchar(1000), field varchar(100), points int,
urlid bigint, anchoruid bigint default -1, anchorurl varchar(5000))

<table>
<thead>
<tr>
<th>wordid</th>
<th>anchorwd</th>
<th>field</th>
<th>points</th>
<th>urlid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>google</td>
<td>inbd</td>
<td>200</td>
<td>3</td>
</tr>
<tr>
<td>73</td>
<td>webmail</td>
<td>inbd</td>
<td>200</td>
<td>5</td>
</tr>
<tr>
<td>74</td>
<td>links</td>
<td>exinbdhead</td>
<td>40</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>anchoruid</th>
<th>anchorurl</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td><a href="http://www.google.com/">http://www.google.com/</a></td>
</tr>
<tr>
<td>-1</td>
<td><a href="http://webmail.uccs.edu/">http://webmail.uccs.edu/</a></td>
</tr>
<tr>
<td>-1</td>
<td><a href="http://webmail.uccs.edu/">http://webmail.uccs.edu/</a></td>
</tr>
</tbody>
</table>

foreign key urlid references URLs(urlid) on update cascade and on delete cascade
Table 5. BASEURLS

(urlid bigint, url varchar(5000), numoflinks int, baseurl varchar(5000) primary key (url, baseurl))

<table>
<thead>
<tr>
<th>urlid</th>
<th>url</th>
<th>numoflinks</th>
<th>baseurl</th>
</tr>
</thead>
<tbody>
<tr>
<td>32542</td>
<td><a href="http://128.198.85.111/Radio/">http://128.198.85.111/Radio/</a></td>
<td>0</td>
<td><a href="http://radio.uccs.edu/">http://radio.uccs.edu/</a></td>
</tr>
<tr>
<td>47461</td>
<td><a href="http://cdpsweb.state.co.us/">http://cdpsweb.state.co.us/</a></td>
<td>24</td>
<td><a href="http://www.uccs.edu/~pusafety/polic2hb.htm">http://www.uccs.edu/~pusafety/polic2hb.htm</a></td>
</tr>
</tbody>
</table>

Table 6. DISPLAYWDS

(urlid bigint, titlewds text, displaywds text)

<table>
<thead>
<tr>
<th>urlid</th>
<th>titlewds</th>
<th>displaywds</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>university of colorado at colorado springs</td>
<td>welcome to the university of colorado at Colorado springs...</td>
</tr>
<tr>
<td>100</td>
<td>academic programs</td>
<td>the university of colorado at colorado springs college of education extended studies program, academic programs, early literacy certificate english as a second...</td>
</tr>
<tr>
<td>133</td>
<td>the university center</td>
<td>university center jeff davis director of operations and management the university center is a campus community based facility for students faculty staff administration and guests..</td>
</tr>
</tbody>
</table>
3.4 Problems Faced and Solutions

Although Web crawling is conceptually easy in that you just follow the links from one site to another, it remains somewhat challenging. Because of the large number of pages, different kinds of unexpected problems may occur and some of them can even crash the program. For example, some links have very long junk urls, which cannot be fetched from the server. For this reason, I added a constraint to the beginning of the program that would drop any urls with a length of more than 100 characters.

Additionally, some pages that are returned by the server do not have the actual content in the page. These pages need to be removed from the index as well.

Extended anchor text is defined by the crawler 10 words before anchor words, and any headers that is within 25 words before the anchortext. But when extended words are anchor words themselves, the rest of the extended words actually describes this link, but not the one before it. Also, sometimes the extended text is irrelevant to the anchor words. Both problems can bring problematic result pages to users. Therefore, I specified in the crawler that the extended anchor words are extracted only if they are in the same tag field.
as that of the extended anchor word that is next to the anchor words, provided they are not anchor words themselves. This is based on the assumption that text words that are in the same field tag usually have a consistent meaning.

Initially, the returned search results were not as expected in terms of their relevance because the weighting system was underrating those keywords in significant fields, and overrating those in less important fields. For example, if a searched word appeared several times in the body that page might be inappropriately ranked higher than if the keyword appeared in the title. To correct this, the weighting system was altered to put more weight on words appearing in the url, title, and so on (recall Table 1). Less weight was given to words appearing in the body of the text. As such, the new weighting system allowed for a greater spread in the rankings, thus providing more accurate results than in the initial testing.

To calculate the PageRank for a page, all of its outbound and inbound links are taken into account. These are links from within the site and links from outside the site. This information is stored in the table baseurls, as mentioned above. According to Google’s algorithm, the page rank of each page depends on the page rank of the pages pointing to it. But we won’t know what page rank those pages have until the pages pointing to them have their page ranks calculated, and so on. This, of course, is circular. But actually it is not as bad as it may seem. Jellinghaus (1995) reminds us with regards to Google that page rank can be calculated with a simple iterative algorithm; as such, page rank corresponds to the principal eigenvector of the web’s normalized link matrix. What that means is that we can go ahead and calculate a page’s page rank without knowing the final value of the page rank of the other pages. This seems strange but,
basically, each time we run the calculation we’re getting a closer estimate of the final value. As such, all we need to do is remember each value we calculate and repeat the calculations many times until the numbers stop changing significantly (http://www.iprcom.com/papers/pagerank/#ex1).
CHAPTER 4
EVALUATING THE EFFICIENCY AND EFFECTIVENESS OF CATSSEARCH

Originally, the present project was designed to compare CatsSearch to the original ht://Dig search engine. However, by the time CatsSearch was completed, UCCS had added a new, Google-powered search engine, ‘UCCS Google,’ to its site. The new UCCS search engine produces much better search results than did the original ht://Dig search engine. Obviously, this change affected the present project. As such, it was decided to evaluate the speed and accuracy of CatsSearch by comparing it with the current UCCS search service, UCCS Google (as explained in 1.2.1) and with Yahoo within the “UCCS.edu” domain. This comparison, within a limited framework, is presented in the rest of this chapter.

4.1 Methodology
CatsSearch, Google, and Yahoo were evaluated on 25 key terms (see Table 8). These terms were taken from Tomaiuolo & Packer (1996), who examined in total 200 subjects relevant to undergraduate curricula. These terms thus seemed relevant to the present study, as the UCCS domain was the focus of the search. A sample screen capture for CatsSearch is provided in Figure 4.
In terms of the statistics, the independent variable in this evaluation was
SEARCH ENGINE (3 levels: CatsSearch, Google, Yahoo). There were three dependent
variables: SEARCH DURATION (i.e., the time it took to complete a particular query on
one of the 25 terms), NUMBER OF “HITS” for each query, and the

Figure 4. Screen capture of CatsSearch with returned results for “capital punishment.”
RELEVANCE/ACCURACY of the searches. Statistical consultation and support was provided by Dr. Kevin Ford, Department of Psychology, Colorado College. All statistics were conducted with the SPSS statistical software (version 11.0.2, McIntosh; see www.SPSS.com). More detailed description of the tests used in the present study can be found elsewhere (see http://www.ats.ucla.edu/stat/spss/; http://www.statsoft.com/textbook/stathome.html; Motulsky, 1999). A standard probability level (p value) of $\alpha = 0.05$ was adopted, meaning that a statistic could only be considered significant with a certainly level of 95%.

4.1.1 Evaluating search times and number of “hits” per query. Search response time is found at the top of each search engine’s search result page. According to Google, search response time is defined as “the total server time to return the search results, measured in seconds” (Google Web APIs Reference, 2005). It is unclear how Yahoo defines search response time, but it most likely does not include network transformation time. For CatsSearch, the search response time is defined as the time (in seconds) from when the servers receives the user input until the server obtains the result pages, but before these results are presented to the client. Because there is frequent variation in search times and in the number of “hits” a query returns, it was necessary to conduct more than one search for each of the 25 terms. Specifically, three separate searches using the same 25 items were performed on three different computers:

1) Mac G4, 1 GHz, System 10.3.9, Internet Explorer 5.2
2) Mac G5, Dual 2.7 GHz, system 10.4.3, Safari 2.0.2
3) HP Pavilion, Pentium II, Windows 98, Netscape 6.2

The results from these three searches (N = 75) were then averaged to provide an estimate of search times and number of “hits” per query for each search engine. These results were evaluated for significant differences among the search engines by running a repeated measures Analysis of Variance (ANOVA). This analysis compares three matched groups by examining three types of variability: between independent variables (i.e., the search engines), between dependent variables (i.e., rankings of the search engines), and random (or residual) variance. It then partitions the total sum-of-squares accordingly and adjusts for the number of groups being compared and the number of observations (i.e., degrees of freedom) to compute an F ratio. The larger the F ratio (usually above 1), the greater the chance that there is a statistical (rather than random) difference among the independent variables.

4.1.2 Evaluating the relevance and accuracy of search results. As outlined above in section 1.2.6, there are many potential ways of evaluating the relevance and accuracy of web engine search results, with advantages and disadvantages to each (Vaughan, 2004). In the present study, it was decided that using human, multi-level judgment to rank the search engine results would be most accurate system. The criteria for rating the relevance/accuracy of search “hits” were adapted from Shang and Li (2002). The slightly modified criteria are as follow:

3 points: Relevant links with information directly related to the query, or with many related links useful to the query.
2 points: *Peripherally relevant links* with information only marginally related to the query. The searched terms are present on the web page (i.e., in the form as one item in a long list), but are not central to the query. An example of a peripherally relevant link would be one that, in response to a query on “date rape”, returns a webpage that discusses some aspect of rape, but makes no mentions of date rape itself, either explicitly or implicitly.

1 point: *Irrelevant links* with information not related to the query, or inactive links that provide error messages (e.g., file not found, 404, forbidden or server not responding, 603 errors). An example of an irrelevant link would be obtaining a web page mentioning the “commonwealth” of nations with regards to Queen Elizabeth in response to the searched term “commonwealth of independent states” (which refers to the 1991 declaration uniting Azerbaijan, Armenia, Belarus and other former Soviet Republics).

To assure reliability in these ratings, only one rater was used. In addition, the rater was asked to rate 20 hits for the term “holistic medicine.” After a delay of two hours, to avoid the effects of working memory, the rater then re-ranked the same 20 hits. A paired sample T-test (for comparing mean differences between two related groups—in this case, between the two sets of rankings) indicated no significant difference between these two trials. Moreover, a Pearson Product-moment correlation, which measures the relationship between two (or more) variables, between the two rating trials indicated a high intra-rater reliability in these ratings ($r = 0.93$); in other words, the two rankings were very closely
matched and consistent with each other. [Note: a correlation coefficient, or r, ranges from -1.0 (a perfect negative correlation) to +1.0 (a perfect positive correlation); an r of 0 means there is no relationship between the variables].

The relevance and accuracy of the search results were obtained by examining the first ten “hits” returned for each of the 25 queries (N = 750). To do this, it was necessary to examine the returned web pages themselves, not just their returned summaries. Without examining the web pages themselves, it was not possible to evaluate the context in which the searched items appeared, and establishing this context was crucial to determining the relevance of the web page to the search.

To avoid possible sequencing effects, search engine order was rotated for each item in Table 1: “Abortion ethics“ (CatsSearch, Google, Yahoo), “Abuse of human rights” (Google, Yahoo, CatsSearch), “Acupuncture” (Yahoo, CatsSearch, Google), and so on. After each “hit” was rated for accuracy on the three-point scale, the results were aggregated to produce an average ranking for the ten (or fewer) web pages for each query—this minimized the problem of missing data if a query returned fewer than ten hits. These aggregated data were then examined with a Friedman $\chi^2$ (1937), a nonparametric test (i.e., a test does not assume a normal distribution of the data) for dependent (i.e., related) samples. Briefly, the Friedman $\chi^2$ compares three or more paired groups by first rank ordering the values in each matched set (i.e., each row of data; in the present data set, the ranking of each searched term). Comparing within matched sets controls for variability between subjects and increases the power of the test. It then sums the ranks of each group (i.e., each column; in the present data, each search engine). If the totals differ substantially among these groups, then there is a greater likelihood that the
differences among the groups are significant and not due to chance (i.e., random effects). In the present study, significant differences would mean that the three search engines actually contributed to the observed differences in the dependent measures (i.e., the relevance rankings).

4.2 Results

It is important to point out that CatsSearch crawled a very small number of web pages compared with Google and Yahoo. Google has crawled over 8 billion web pages, and Yahoo has crawled 20 billion pages, whereas CatSearch crawled only 50,000 pages. This difference affected return time, number of returned pages, and the page quality of search results. As such, the following comparisons are somewhat limited. Even then, CatsSearch did manage to return relevant pages (rankings of 2 or 3) that did not show up in the first ten pages with Google or Yahoo. It should also be noted that there was considerable overlap in the returned pages across the three search engines.

4.2.1 Search times and number of “hits” per query. There were significant differences among the three search engines in terms of the search time/query, F(2, 55) = 69.37, p < 0.001, and the number of returned hits/query, F(2, 148) = 20.95, p < 0.001. [Note: because the F statistic here is considerably above 1, and the p value is less than 0.001 (well below the set \( \alpha \) of 0.05), the observed difference among the search engines is probably a real, not random, difference. Numbers within parentheses represent the
degrees of freedom for the test (number of groups minus 1, sample size minus the number of groups).]

The repeated measures ANOVA in itself, does not tell you where these differences among the independent variables (i.e., the search engines) lie. To specify which search engine differs from which, one needs to conduct a post-hoc analysis, which SPSS does for you. In the present sample, the post-hoc analyses for time/query and hits/query indicated significant differences between each pairwise search engine comparison (e.g., CatsSearch-Google, CatsSearch-Yahoo, Google-Yahoo). Thus, each search engine was, in fact, significantly different from the other two search engines in terms of time/query and hits/query. Descriptively, within the limitations of the present comparisons, in terms of the average search time for each query, CatsSearch was the fastest, and Yahoo was the slowest (see Figure 5).

In terms of the average number of returned hits for each query, Google had the highest number, and CatsSearch provided the fewest (see Figure 6). Both Google and CatsSearch were very consistent in the number of hits they returned; Yahoo, however, exhibited much more variation in the number of returned pages.
Figure 5. Average search times (in seconds) for the 25 queries across each search engine. Note that CatsSearch was the fastest among the three search engines for these particular test items within the UCCS domain. Error bars represent standard deviations.
Figure 6. Average number of hits for the 25 queries across each search engine. Note that Google returned the highest number of hits, and CatsSearch returned the lowest number. These results obtain only for these particular test items within the UCCS domain. Error bars represent standard deviations.

A break down of individually searched items within the UCCS domain is provided in Table 8. CatsSearch failed to return any hits for 4 of the queries, whereas Google and Yahoo failed to return hits only on 2 queries. For all three search engines combined, the search queries that returned the fewest number of hits were “Lindane toxicity” (n = 0) and “injection molding” (n = 1). The highest number of hits was returned for “date rape” (n = 413) and “environmental marketing” (n = 380).
Table 8: Average search time and number of “hits” for each of the 25 items searched within the UCCS domain

<table>
<thead>
<tr>
<th>Searched Item</th>
<th>CatsSearch</th>
<th>Google</th>
<th>Yahoo</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abortion ethics</td>
<td>---- (0)</td>
<td>0.30 (11)</td>
<td>0.52 (2)</td>
</tr>
<tr>
<td>Abuse of human rights</td>
<td>0.30 (26)</td>
<td>0.34 (116)</td>
<td>0.96 (53)</td>
</tr>
<tr>
<td>Acupuncture</td>
<td>0.18 (2)</td>
<td>0.38 (10)</td>
<td>0.41 (4)</td>
</tr>
<tr>
<td><em>Beowulf</em> criticism</td>
<td>0.17 (1)</td>
<td>0.40 (5)</td>
<td>0.53 (1)</td>
</tr>
<tr>
<td>Capital punishment</td>
<td>0.20 (15)</td>
<td>0.47 (65)</td>
<td>0.57 (23)</td>
</tr>
<tr>
<td>Caribbean history</td>
<td>0.24 (28)</td>
<td>0.42 (55)</td>
<td>0.55 (33)</td>
</tr>
<tr>
<td>Chemical warfare</td>
<td>0.16 (15)</td>
<td>0.43 (42)</td>
<td>0.60 (22)</td>
</tr>
<tr>
<td>Commonwealth of Independent States</td>
<td>0.24 (3)</td>
<td>0.38 (11)</td>
<td>0.69 (5)</td>
</tr>
<tr>
<td>Date rape</td>
<td>0.23 (18)</td>
<td>0.30 (331)</td>
<td>0.86 (65)</td>
</tr>
<tr>
<td>Effect of divorce on children</td>
<td>0.24 (6)</td>
<td>0.35 (33)</td>
<td>0.58 (9)</td>
</tr>
<tr>
<td>Environmental marketing</td>
<td>0.22 (40)</td>
<td>0.29 (181)</td>
<td>0.76 (159)</td>
</tr>
<tr>
<td>Flat tax</td>
<td>0.18 (4)</td>
<td>0.43 (42)</td>
<td>0.63 (23)</td>
</tr>
<tr>
<td>Forensic accounting</td>
<td>0.24 (8)</td>
<td>0.51 (43)</td>
<td>0.45 (21)</td>
</tr>
<tr>
<td>Germany reunification</td>
<td>0.14 (1)</td>
<td>0.31 (3)</td>
<td>0.45 (3)</td>
</tr>
<tr>
<td>Holistic medicine</td>
<td>0.16 (14)</td>
<td>0.30 (49)</td>
<td>0.60 (27)</td>
</tr>
<tr>
<td>Impressionism</td>
<td>0.17 (3)</td>
<td>0.33 (18)</td>
<td>0.48 (7)</td>
</tr>
<tr>
<td>In-vitro fertilization</td>
<td>---- (0)</td>
<td>0.44 (4)</td>
<td>0.66 (2)</td>
</tr>
<tr>
<td>Injection molding</td>
<td>---- (0)</td>
<td>0.23 (1)</td>
<td>---- (0)</td>
</tr>
<tr>
<td>Insurance industry</td>
<td>0.18 (55)</td>
<td>0.28 (126)</td>
<td>0.63 (157)</td>
</tr>
<tr>
<td>International importing/exporting</td>
<td>0.21 (2)</td>
<td>---- (0)</td>
<td>0.49 (4)</td>
</tr>
<tr>
<td>John Singer Sargent</td>
<td>---- (0)</td>
<td>0.30 (2)</td>
<td>0.33 (2)</td>
</tr>
</tbody>
</table>
Table 1: Cont.

<table>
<thead>
<tr>
<th></th>
<th>Time (seconds)</th>
<th>Hits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Legalizing marijuana</td>
<td>0.20 (2)</td>
<td>0.33 (4)</td>
</tr>
<tr>
<td>Lindane toxicity</td>
<td>---- (0)</td>
<td>---- (0)</td>
</tr>
<tr>
<td>LSAT</td>
<td>0.15 (26)</td>
<td>0.38 (52)</td>
</tr>
<tr>
<td>Malcolm X</td>
<td>0.21 (11)</td>
<td>0.37 (33)</td>
</tr>
</tbody>
</table>

Data are presented as follows: time in seconds (# of hits) averaged across the three trials.

4.2.2 Relevance and accuracy of search results. Although there was variation in the average ranking among the three search engines, there were no significant differences, Friedman $\chi^2(2, 19) = 3.09$, p = 0.214, which is not significant [Note: the $\chi^2$ value here is similar to the F statistic discussed above, and is relatively small; the p value is well above the set $\alpha$ level of 0.05]. As such, the search engine did not differ in a statistically significant sense in terms of the accuracy of the pages they returned. Descriptively, Yahoo and CatsSearch tended to be more relevant/accurate in their hits than did Google for the 25 test items (see Figure 7). One possible reason for the lowered average rankings in Google may have been that, based on the rater’s recall, a disproportionate number of returned “hits” from Google were inactive links. Quantitative data on this observation are not available, however.
Figure 7. Average ranking (in a 3 point scale) of the accuracy/relevance of web “hits” based on the first 10 “hits” for the 25 queries. Note that Google tended to have the lowest accuracy/relevance ratings in its returns for these items, with CatsSearch and Yahoo being about equal. Note these results obtained only for searches within the UCCS domain. Error bars represent standard deviations.

4.3 Discussion

The present study is very limited in scope because (1) only a small selection of search items (N = 25) were used to test the search engines, (2) because searches were only conducted within the UCCS domain, and (3) because CatsSearch searched a much more limited number of web pages than did the other Google or Yahoo. Within these limitations, the present results indicate that CatsSearch was roughly equivalent to Google and Yahoo in searching web pages within the UCCS domain. On the whole, Google
tended to return more pages than either CatsSearch or Yahoo, but was somewhat less accurate in terms of these returns, perhaps because it tended to return a higher number of inactive links. In previous studies (Vaughan, 2004), Google also tended to retrieve more pages than other search engines. CatsSearch and Google returned a consistent number of hits, as has been documented previously for Google (Vaughan, 2004). Yahoo, however, tended to exhibit much more variation in the number of hits returned with each search of the same queried item.

In terms of speed, CatsSearch was consistently the fastest of the search engines (recall Table 8) but, again, this is perhaps due to the fact that CatsSearch only searched a relatively small number of web pages. In the future, CatsSearch could be extended to include these other file types. Within the confines of the study, CatsSearch also proved to be roughly as accurate as Yahoo, although there were no significant differences in the relevancy/accuracy rankings among the three search engines. However, CatsSearch did, occasionally, return relevant pages that did not appear in the first ten returns of Yahoo or Google.
CHAPTER 5

CONCLUSION

The present study outlined the development and testing of a new search engine, CatsSearch, for the UCCS domain. CatsSearch crawled 53,299 pages, and indexed over 8482581 words. A total of 11,789,595 table rows were created. The crawling process itself took about a week. Adding the InboundWords to the IndexedWords took about 24 hours. Pagerank calculation took 24 hours. Notice these statistics assume that more than one program is usually running at the same time; in addition, these numbers were subject to the slowness of remote connections (e.g., modem) to the UCCS servers.

Although the study was originally designed to improve the old UCCS search engine (ht://Dig), this was not possible because UCCS switched to a Google-powered system part way through the present project. As a consequence, I could no longer compare CatsSearch to ht://Dig. Instead, the present study compared CatsSearch with Google and Yahoo within the UCCS domain on 25 test items. Within the limitations of the present study, the results suggest that CatsSearch is faster than Google and Yahoo in terms of retrieval times and, although much more limited the number of returned pages, roughly as accurate as these well-established search engines for the 25 items used for testing.
Several issues remain to be addressed regarding the present project and potential future extensions of CatsSearch. These are outlined briefly below:

5.1 Problems and Limitations with CatsSearch

Although CatsSearch provided some relatively good results, there are still several problems and limitation:

- **Depth-first crawling**: The current search engine employed depth-first crawling, whereby the crawler followed a particular link until all subsequent links were exhausted. This is not a well-balanced way of crawling because page links are unevenly followed. This problem directly affects the value of the PageRank calculation since PageRank is calculated based each page’s inbound and outgoing links. Another problem with depth-first crawling is that it could potentially lose high quality pages since the crawler is not guided to crawl the best pages according (as mentioned in Section 1.2.3).

- **Limited domains and file types**: Due to the limited coverage of CatsSearch, only pages within the UCCS domain and only pages that were of .(s)htm(l) type were crawled. This definitely limited performance.

- **Abuse of keywords in web pages**: Some top result pages returned by CatsSearch may have turned out to be of low quality because some
webmasters put keywords within the meta tag and/or within other places repeatedly in order to obtain a high rank.

- **Abuse of links in web page**: As with the abuse of keywords, links are manipulated by some webmasters with the purpose of gaining a higher PageRank.

- **Tokeparser problems**: The current search engine used the HTTP::tokeparser module to extract text words from each tag in web pages. But sometimes it could not obtain the expected tag field for corresponding text words because tokeparser lacks ways of dealing with nested tags. For example, a page with the url `http://cs.uccs.edu/` has a link `<a href="http://cs.eas.uccs.edu/"><b>computer science</b></a>`. The tokeparser simply returns text ‘computer science’ as within the `<b>` tag instead of the `<a>` tag, and thus the page with the `http://cs.uccs.edu/` loses an anchor text and the page with the url `http://cs.eas.uccs.edu/` loses an inbound anchor text.

- **Phrase search**: Phrase search in the current search engine took too long to return results if the returned results involved many pages.

- **PageRank application**: Because the PageRank calculation is based on each page’s inbound and outgoing links, and because CatsSearch uses in-depth crawling, PageRank values are not as accurate as they could be.

- **Search features limitation**: At present, CatsSearch supports only a simple search, and the Boolean search ‘AND/OR’, which is an obvious limitation.
5.2 Suggested Future Work to Improve CatsSearch

The following are suggestions for improving CatsSearch:

- **Use a distributed computer system to crawl pages on a larger scale:**
  The current search engine crawled only 50,000 pages. To improve the effectiveness of performance, substantially more pages need to be crawled. To efficiently crawl such large number of pages, a distributed computer system needs to be used to distribute workload over different computers.

- **Use more efficient crawling methods:** Because of the tremendous number of pages on the Web and because search engines are limited in the number of pages they can crawl (recall Section 1.2.3), it would be better to use a more efficient crawling method to crawl high quality pages (such as the breadth-first method).

- **Reading robot.txt:** It is suggested that the crawler read the robot.txt of each web site so that it can avoid crawling pages that are forbidden.

- **Use a compression and caching techniques for indexing:** Even though CatsSearch can answer a user’s query in a reasonable time, as the search engine becomes larger, it will need to employ caching and index compression techniques to improve its search efficiency. Two- or Three-level caching, and lossy compression are appropriate are possible solutions (recall Sections 1.2.4 and 1.2.6).
• **More ranking factors**: Additional weighting factors can be considered for ranking web pages. For example, taking word proximity into account when searching will provide better search results for multiple word searches.

• **Provide full-fledged search features**: As CatsSearch becomes more comprehensive in its search capabilities, it would be possible to add more search features and/or an “advanced” search option.

• **Use data mining**: Another issue of relevance is data mining, which is capable of analyzing patterns in masses of data; analyzing such patterns could result in a more efficient search engine.

### 5.3. Thoughts about Evaluating Accuracy and Relevance

Issues critical to accuracy and relevance have been addressed above and elsewhere (Vaughan, 2004), but a few concerns remain. The present study relied on the judgment of only one individual in ranking the relevance of web pages. This is clearly a limitation for several reasons:

• One individual cannot have sufficient knowledge in enough topics to accurately rank a large variety of web page searches—it may be best to have “experts” judge relevance within their own area of expertise (Voorhees & Harman, 2001).
• It would be better to have a group of individuals come to a mutual agreement on relevance rather than to rely on a single individual (Vaughan, 2004).

• To remove any potential bias to a particular search engine, all rating should be done blind as to the search engine.

Incorporating all of these methodological changes would clearly allow one to make stronger claims about the accuracy and relevance of search engines results, but such changes were well beyond scope of the present project.
REFERENCES


http://www.ats.ucla.edu/stat/spss/ [retrieved November 15, 2005].


https://services.google.com/pss_faq.html#6 [retrieved November 18, 2005].


Retrieved November 15, 2005, from