DISCLAIMER OF QUALITY

Due to the condition of the original material, there are unavoidable flaws in this reproduction. We have made every effort possible to provide you with the best copy available. If you are dissatisfied with this product and find it unusable, please contact Document Services as soon as possible.

Thank you.

Pages are missing from the original document.

missing 3, 5, 6, 9
Development of Genre and Function Types for Web Page Classification and Rating

by

Beethoven Cheng

Submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degrees of

Bachelor of Science in Electrical Engineering and Computer Science

and

Master of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1996

© 1996 Beethoven Cheng. All Rights Reserved.

The author hereby grants to M.I.T. permission to reproduce and to distribute copies of this thesis document in whole or in part, and to grant others the right to do so.

Author ..................................................

Electrical Engineering and Computer Science

May 6, 1996

Certified by ..............................................

Wanda J. Orlikowski
Thesis Supervisor

Certified by ..............................................

Joanne Yates
Thesis Supervisor

Accepted by ..............................................

Professor Frederic R. Morgenthaler
Professor

Electrical Engineering and Computer Science

JUN 11 1996

LIBRARIES
Development of Genre and Function Types for Web Page Classification and Rating

by

Beethoven Cheng

Submitted to the Department of Electrical Engineering and Computer Science on April 15, 1996, in partial fulfillment of the requirements for the degree of Master of Engineering in Computer Engineering

Though the World-Wide Web (WWW) has piqued the imagination of corporation and marketers, little is known about the actual popularity of specific web pages. Instead, web site managers have relied on internal counts of hit rates and outside facilities like Webcrawler's and CMU's Lycos to bolster claims that their pages are visited often. However, these methods, while good to a first approximation, do not take into account a web page's content. This thesis introduces a classification scheme for web pages which forms a normative basis for determining a web page's popularity.

Thesis Supervisor: Wanda Orlikowski
Title: Associate Professor, Sloan School of Management

Thesis Supervisor: Joanne Yates
Title: Associate Professor, Sloan School of Management
DISCLAIMER OF QUALITY

Due to the condition of the original material, there are unavoidable flaws in this reproduction. We have made every effort possible to provide you with the best copy available. If you are dissatisfied with this product and find it unusable, please contact Document Services as soon as possible.

Thank you.

Pages are missing from the original document.

missing 3, 5, 6, 9
Table of Contents

1 Introduction ........................................................................................................................................ 10
  1.1 The World Wide Web .................................................................................................................. 10
  1.2 Project Purpose and Scope ......................................................................................................... 12
  1.3 Project Development .................................................................................................................. 12
2 Background ........................................................................................................................................ 14
  2.1 Current Popularity Metrics ........................................................................................................ 14
  2.2 The Need for Normative Guidelines for Making Automated Comparisons ......................... 25
3 A Classification Strategy .................................................................................................................. 28
  3.1 A New Method for Classification .............................................................................................. 28
  3.2 The Concept of a Genre ............................................................................................................ 29
  3.3 Previous Work .......................................................................................................................... 29
  3.4 Classification Strategies and Popularity .................................................................................... 30
  3.5 The Concept of a Function Type ............................................................................................... 31
4 Methodology ...................................................................................................................................... 34
  4.1 Reason and Motivation .............................................................................................................. 34
  4.2 Overview of Methodology ........................................................................................................ 34
  4.3 Software Design ....................................................................................................................... 37
5 Results ............................................................................................................................................... 42
  5.1 On Selecting and Cleaning the Sample ...................................................................................... 42
  5.2 Collection of Statistics .............................................................................................................. 42
  5.3 Comparison of Pages ................................................................................................................ 44
6 Overall Remarks ................................................................................................................................ 50
  6.1 Satisfaction of Result ............................................................................................................... 50
  6.2 Implications for Web Page Design ........................................................................................... 50
  6.3 Improvements and Extensions to the Categorization Scheme and its Dependent Algorithms 51
  6.4 Improvements in Code Efficiency ............................................................................................. 54
  6.5 The Need for Complete Coding Automation ......................................................................... 55
  6.6 The Need for HTML Stability ................................................................................................. 56
  6.7 Restrictions of Structure-based Classification Schemes ....................................................... 56
DISCLAIMER OF QUALITY

Due to the condition of the original material, there are unavoidable flaws in this reproduction. We have made every effort possible to provide you with the best copy available. If you are dissatisfied with this product and find it unusable, please contact Document Services as soon as possible.

Thank you.

Pages are missing from the original document.

missing 3, 5, 6, 9
DISCLAIMER OF QUALITY

Due to the condition of the original material, there are unavoidable flaws in this reproduction. We have made every effort possible to provide you with the best copy available. If you are dissatisfied with this product and find it unusable, please contact Document Services as soon as possible.

Thank you.

Pages are missing from the original document.

missing 3, 5, 6, 9
List of Figures

Figure 1.1: Section of statistics page belonging to PFSNet, a home page for alumni of Penang Free School, Malaysia........................................................ ................................................ 15
Figure 1.2: Icons of Internet Audit and Interse. Internet Audit is an auditing system which relies on the icon being fetched every time the page being audited is hit.................................................. 17
Figure 2.1: Icons of Internet Audit and Interse. Internet Audit is an auditing system which relies on the icon being fetched every time the page being audited is hit ........................................ 21
Figure 2.4: Page of Point Communications showing links to lists of pages which rated highest on each of its three criteria................................................................. 23
Figure 4.1: Graphic representation of spider traversing html links ........................................ 39
List of Tables

Table 3.1: Function Types and Their Specifications 38
Table 5.1: Comparison of Means of Form Features Between Lycos and the Random Set. *p<0.05; **p<0.01 49
Table 5.2: Normalized Features Derived from Features of Referential Lycos and the Random Set Summarized in Previous Table. *p < 0.05, **p < 0.01 50
Table 5.3: Breakdown of the 146 pages from Lycos and from the random set 51
Table 5.4: Comparison of Means of Form Features Between Referential Lycos and Random Set Pages. *p < 0.05, **p < 0.01 53
Table 5.5: Normalized Features Derived from Features of Referential Lycos and The Random Set Summarized in Previous Table. *p < 0.05, **p < 0.01 53
Table 5.6: Comparison of Means of Form Features Between Informational Lycos and Random Set Pages. *p < 0.05, **p < 0.01 54
Table 5.7: Normalized Features Derived from Form Features of Informational Lycos and the Random Set Summarized in the Previous Table. *p < 0.05, *p < 0.01 54
DISCLAIMER OF QUALITY

Due to the condition of the original material, there are unavoidable flaws in this reproduction. We have made every effort possible to provide you with the best copy available. If you are dissatisfied with this product and find it unusable, please contact Document Services as soon as possible.

Thank you.

Pages are missing from the original document.

missing 3,5,6,9
Chapter 1

Introduction

1.1 The World Wide Web

The World Wide Web (WWW) has piqued the imagination of corporations and marketers more than any other communications vehicle since television. Electronically linking an estimated 30 to 50 million users [1][4] to a vast body of information, the WWW is causing corporations to rethink their marketing and sales channels and their internal and external communications strategies [10]. The WWW is not only the quintessential corporate bulletin board, but a means to share documents, applications, and information between offices, between departments, and around the world.

The popularity enjoyed by the WWW has been marked by rapid growth. With a size of around 30 million pages, a growth rate estimated at 300,000 new pages every day and a doubling period of 6 months [2][4], the WWW is now seen as a serious media platform, with potential to someday rival TV and radio.

However, in contrast to the apparent popularity of the WWW, little is known about the actual popularity of specific web pages. Few independent studies have been conducted on determining usage of specific pages. Instead, web site owners often maintain statistical daemons which track the hit profiles of pages resident on their web site. These numbers are then attached to the web pages as proof of the page's reader rate.

Such information does give some indication of a page's popularity. A hit profile can be used to construct the usage of a page over a sample slice of time. Moreover, it may provide
details on where the hits are coming from and how the hits are spaced. Thus, a hit profile provides a summary of a page’s usage.

On the other hand, a hit count does not tell how popular a page is relative to other pages in the world. To determine this, a global comparison scheme needs to be instituted which would compare the hit profiles of all the known web pages to each other.

Such a comparison scheme is a difficult undertaking since it requires that all web page owners voluntarily contribute their pages’ access information. In the case of local statistical daemons, the information gathered is usually maintained in local privileged memory, accessible only to system administrators of the web site. If a page owner wants to publicize his page’s access data, he would have to ask an administrator to grant him access to the information or set things up for him so that the information can be automatically presented to the outside community. For a global comparison scheme to be instituted, one would need to enforce a hit count reporting standard which would be accepted by a large majority of the administrators and web page owners alike, a monumental task to say the least.

Methods of cataloguing usage involving external auditing systems have been proposed and implemented which deal to some extent with this problem. These systems are maintained for collecting hit rate data over the Internet and work by tricking browsers which access an audited page into fetching a specific icon from the auditing system. Every time this icon is fetched, the auditing system credits a hit to the particular audited page and updates a hit profile database.

Because such a system requires a user to register and add the icon to his/her page, requiring them to accede to disclosure standards becomes part of the registration process. Moreover, because the access data is maintained centrally in the auditing machine, comparisons of hit profiles are easier and more flexible.
However, auditing systems can become victims of their own success. Because information is centrally kept, it is very easy for a popular audit system to become bogged down by browsers requesting icons. Thus, hit counts maintained on an overloaded auditing system are bound to be underestimates. A similar consequence is that hit profiles maintained on these systems are often more depressed during web peak usage times even if usage of that page actually increased at that time.

Furthermore, even if a global comparison scheme were possible, a comparison makes sense only if the forms of information presented in the pages being compared (i.e., news articles, graphics, etc.) are comparable as well. Care needs to be made to ensure that only pages which serve the same purpose are compared to one another. Again, because a hit does not reveal why a certain page is being accessed, one cannot determine strictly from hit profiles if two pages serve the same purpose.

1.2 Project Purpose and Scope

This thesis proposes that the primary reason why determining a page’s popularity is difficult is because the common comparison techniques either fail to take context (i.e., the page’s usage) into account or are too slow to be feasible; comparing web pages can be done most efficiently if one determines by observing a page’s structure what purpose that page serves.

Given these points, the thesis proposes a web page classification scheme which elicits usage information from the physical structure of a page. The scheme and an implementation will be discussed, and its performance demonstrated and critiqued.

1.3 Project Development

This thesis breaks the topic into six parts.
First, it will discuss in detail the popular audit systems in use today and the ways in which they fail to properly measure a page’s popularity. It will show that these problems can be corrected by taking into account a page’s form and function, and afterwards, developing a set of normative guidelines through which web pages can be classified and compared.

Second, an outline for a classification scheme will be introduced. This section will begin by discussing the technological backgrounds for the scheme, including genres and function types, and previous work done on classifying web pages. It will continue to outline how the scheme partitions web pages according to a concept called web page function types. Finally, it will demonstrate how the scheme can avoid the difficulties encountered by other popularity measurement scheme.

Third, the methodology for implementing and testing this classification scheme will be presented.

Results of field runs, and a discussion and interpretation of these results will be made in section four. Field runs involved coding the Lycos 250, a set of popular pages listed by the Lycos web search engine from Carnegie Mellon University, along with a similar number of randomly chosen pages.

The thesis will then conclude by discussing possible means for extending the work and suggesting some areas where the technology, scheme, and automatic coding can be applied.
Chapter 2

Background

2.1 Current Popularity Metrics

Currently, there are two predominant metrics for determining a page's popularity - a page's hit count and link count. A third less prominent method that uses human perception to gauge a page's popularity - a subjective metric - is also used and will be discussed in this section.

A web page's hit count is the number of times the page has been accessed over a period of time, while its link count refers to the number of pages which maintain pointers to that page, the assumption being that the link count indicates how many users intend to use the page regularly. Each method assumes that any instance of the statistic they are interested in is of equal importance (i.e., the former that any two hits are of equal importance and the latter that any two links are of equal importance). Thus, each collection method suffers from unique biases resulting from instances when the assumptions fail to hold, biases which search engines adopting a particular method try to combat. This section will discuss these methods in detail.

Hit Counts

The hit count metric involves associating a counting tool with the page, primarily a counter software residing on some web site on the network. The counting tool is activated whenever the page of interest is accessed. It then records relevant page hit information - time of the hit, the requesting site of the hit, or the uniqueness of the hit in relation to other hits made recently - onto a hit log. After a sufficiently long period of time, the data in the log is analyzed and a number of statistics such as time of hits, spatial and temporal relation of hits are gathered.
Systems which employ this method of data collection, called 'counters' in this paper, come in two flavors, one implemented together with the web site (local counters) and another implemented by an external company (external counters) and attached to the page by the page owner/author.

Local counters monitor the usage of pages which are on the server/site on which they run. Such counters come as part of the web server software. Every time a request is made to the site for a particular page, the counter is informed of which page is being accessed, at what time and where the access request is originating from. These counters in turn maintain a usage log which they continually update. Tools which quantify usage are then applied to these logs, producing tables and usage statistics which can then be published on the web.

Usage logs are files which are usually located in privileged/root partitions of the site disk. Part of the reason for locating them there is historical, since almost all machine statistics are kept in root disk space. A more important reason for locating them there is the fact that the server software is usually maintained by a system or site administrator, an individual who owns or is responsible for the maintenance and operation of the site. As a consequence, web usage information is normally controlled only by a few individuals. Pages which are tracked by the counters are usually maintained by the same people who manage the site. Others who own pages but do not maintain the site obtain their pages' usage data by, depending on the site policy, either requesting the administrators to release the pertinent information to them on a regular basis, or using software made available by the administrator(s) for specifically that purpose.

An example of such a system can be found at [13].

Figure 2.1 presents the top page of [13] as it was observed on March 20, 1996.
In contrast to local counters, external counters track pages belonging to individuals unrelated to the ones who maintain these counters. Examples include systems established by Interse (http://www.interse.com) and the Internet Audit System (http://www.iaudit.com), two commercial auditing companies on the World Wide Web (see figure 2.2). Such systems work
by having a page owner include in his/her page an object (a gif picture, for example) provided by the counter’s site. This object is then fetched whenever the page is accessed, in turn triggering a counting system, similar to the one employed for local counters, which notes down what page was accessed, when it was accessed and by whom (i.e., which machine).

**Figure 2.2:** Icons of Internet Audit and Interse. Internet Audit is an auditing system which relies on the icon being fetched every time the page being audited is hit

![Icons](image)

**Problems with Hit Counts**

For the hit count metric, the central assumption is that hit profiles are strongly indicative of a page’s popularity; the greater the number of hits, the more popular a page is. While at first glance this assumption may be reasonable, further examination reveals several problems in this method.

Maintaining a uniform authority for counting hits is difficult. The difficulty exists because of how the WWW was formed; the web is a distributed system based on the client-server paradigm of computing. As such, it operates with no centralized control. Instead, machines interact with each other through the use of agreed upon protocols.

This lack of centralized control makes it hard to verify the correctness of data reported by machines. Credibility and reliability of data become difficult to prove. Also, determining the status of a machine is also problematic. For example, one cannot tell if a machine is just taking awhile to respond or if it is actually down.
The statistician has two ways of collecting data. First, he can choose to rely on the site managers to collect data on their own, by maintaining a local counting system. The issue would then be one of credibility (i.e., how much confidence can the statistician put on these managers and what do these managers need to do to gain the statistician’s trust). Might a manager not inflate his result by, perhaps, changing the counting software so that it double counts hits? Or perhaps the manager might write a script which automatically pings every few seconds the page whose hit count is to be inflated?

The second method is to audit pages remotely by adding a resource to the page of interest. This resource, such as a graphics icon, is considered part of the page and, thus, will be automatically fetched together with the page, triggering a counting tool which has been previously coupled with that icon.

While this strategy works pretty well and does not put the burden of proof solely on the web site manager, it assumes properties about the web page’s site and the auditing site which may not always hold true.

First and foremost, it assumes that the auditor web site is never congested (i.e., every time a page that is being ‘audited’ is pinged, the associated ping which is sent to the auditor is not lost). This assumption is true if the audit site is on a sufficiently high bandwidth network, if the machine it is running on is fast enough, and lastly, if the service never audits too many pages at once. If any of these assumptions fails, the auditor will likely underestimate a page’s hit count, especially when most of the hits striking a page occur during peak hours of usage.

Secondly, it assumes that the people using the web page fetch it using a graphical browser located on a site whose bandwidth is sufficient enough so that fetching pictures is not tedious. Browsers like Mosaic and Netscape are graphical in nature, but Lynx, another popular browser, is text based. Furthermore, many sites today do not reside on fat ethernet trunks. A graphics-intensive page resident on such a site may take a long time to load and the user may
choose to not download the icon. Again, the immediate effect is that audits become underestimated.

Even when data for every page can be obtained in an unbiased, completely efficient manner, a separate problem arises; hit profiles, on their own, provide very little information other than when and from where a page was accessed. Useful information can only be arrived at if pages are compared to one another.

Unfortunately, hit profiles vary over different classes of pages. For example, newspaperlike web pages (i.e., ESPN Sportszone, Clarinet News) tend to be hit with bursts around the time a major news event occurs. One might recall the articles concerning the web pages which showed the Jupiter collision pictures last year. The number of hits sustained by a page carrying such an article was correlated with the size of the splash made by the space debris striking the surface of Jupiter. The load on the Jupiter servers was particularly high when chunks Q1 and Q2 struck. Similarly, a web site maintained by a research scientist was slowed down immensely by a surge in hits during a six month period because he had left online pictures of a prank/experiment he had conducted which had been written about by Dave Barry, a nationally syndicated humor columnist.

At the other extreme, personal web pages are struck at random. Hits would result when friends visited the page. Since a user's page is unlikely to change often, these friends are unlikely to visit it often either. Moreover, the distribution of the hits over time would appear to be even, since friends may visit his page whenever they feel like it.

Somewhere in between are Internet databases. These pages have hit profiles which are more bursty than those of personal home pages but not as bursty as those of newspaperlike webpages. The burstiness results from users submitting in a short span of time a number of related search requests. Thus, hits often arrive in clusters. However, because users tend
to look up databases independently, these request bursts tend to be more evenly spread out than newspaper-like pages.

Comparing the hit profile of a newsgroup page to that of an Internet database is not as meaningful as a comparison of pages of a similar hit profile. A web page on baseball will have a smooth hit profile during the winter months when no baseball is played in this country than around the time of the World Series, when baseball fever hits it peak. On the other hand, an Internet movie database’s usage will peak during the summers and the winters when many blockbuster movies are released. What then can we conclude by a study that shows that www.baseball.com experienced only 10% of the hits the Internet movie database saw between December and February?

As we have seen, hit counts are only good as first approximations to assessing a page’s usage. To better understand a page’s popularity, related information needs to be taken into account, including data which accompany hit counts - for example, the locations, timing and frequency of the hits. Moreover, information not so readily attainable is also helpful. Part of this project will be to articulate what these other data are.

**Link Counts**

The second method of measuring page popularity involves determining the directed graph representation of the web and counting the in-edge number for every node. The directed graph representation of the WWW models web pages as nodes and links as directed edges. Thus, a page’s popularity is determined by the number of edges which point at the node corresponding to the page. This metric is based on the assumption that there is a strong correlation between the number of links pointing at a page and the number of users who access that page.
This link count metric is implemented through the use of web spiders, a class of software robots for the WWW which traverse the web by going from page to page through links. A spider crawls through the web and catalogues all the unique pages it has seen as well as how many times it has seen a particular page, traversing them in some order, often depth first or breadth first search. In depth-first search (DFS), a spider starts its search from a root page and tries to get away from that root page as quickly as possible, backtracking on its path only when it can go no further. In breadth-first search (BFS) the spider starts its search at a root page and grows a concentric search zone which it expands evenly by one link each time. Examples of this collection scheme include Lycos[11] at CMU and Webcrawler[12]. (see fig. 2.3)

**Figure 2.3:** Lycos’ Logo: Lycos claims to have catalogued over 91% of the web.

---

**Problems with Link Counts**

Three assumptions are key for the link count metric. First, the mechanism which explores the web has to know a good percentage of the existing web pages in the world. Second, the web pages need to be fairly static since an exploration of the web is usually done over a period of several months. If web pages changed very often, snapshots of the web would no longer be valid by the time the spider completes its search. Lastly, all links are of equal value (i.e., each link provides approximately the same number of hits to a page). These assumptions are often violated. When they fail to hold, errors in the link count can be introduced, thus making this metric unreliable.
The first two assumptions hold true depending on the performance of the spider and where the spider resides on the WWW. A spider needs to be fast and have good memory in order to traverse the web efficiently and quickly. Moreover, every page needs to be reachable from where the spider resides; otherwise, not all the pages in the world may be explored. These performance and structure requirements for the spider and the web at large are difficult to meet since the internet is inherently chaotic. In fact, the internet makes no guarantees on routing delay time. Thus, a page access may forever remain uncompleted if the network surrounding the originating site of the page access is sufficiently bogged down with net traffic. Furthermore, there are sites in the web which are effectively spider traps; these sites automatically generate new web pages every time an access is detected. The spider then regards these new pages as unexplored territory, starting an unending cycle from which the spider does not recover.

The third assumption has several problems. First, a good number of pages refer to themselves directly. A spider must be smart enough to discard self-referential links since these links do not add to a page's popularity but instead exist solely to help users retrace their steps. Second, in general certain links are more important than others. A link to a page from a popular site arguably adds more to a page's popularity/visitation rate than a link from a site visited only by a handful of students each year. Three, links may be dynamically established. A link to a page from a source page that lasts only a second counts much less than a link from a permanent database. Examples of such links include those generated by Lycos when it returns a link count result, or those generated by Infoseek. Fourth, not all links to a page exist on the web. In particular, with the introduction of bookmarks by Netscape, many users no longer keep track of their favorite pages through a hotlist on their homepage.

Miscellaneous Difficulties with Hit Counts and Link Counts
Several other problems affect both principal data collection schemes:

1. Certain web pages maintain state information in their links. For example, the Human genome project maintains dynamic webpages whose links contain information about when they were generated. These sites would appear to a robot to have an almost infinite number of pages to explore since the site never identifies a page by the same html address.

2. Certain web pages do not use any of the recognized audit systems because their administrators have their own audit system, do not want the pages to be audited, or audit the pages using separate auditing schemes other than the ones studied.

3. Pages may be linked to certain web browsers as 'default' web pages for the browsers. This contributes to the high hit counts of Netscape's homepage and, at home, to the high hit counts of MIT's SIPB homepage.

**Subjective Metrics**

A third method of determining web pages involves actually visiting web pages, subjectively judging the quality of these pages (in terms of presentation, layout, etc.) and using the judgements as a means of measuring how often a page will be accessed.

This method is employed by Point Communications, now owned by Lycos, Inc. It critiques pages along 3 criteria - content, presentation and experience. Each page visited is given a score for each criterion ranging between 0 and 40.

**Figure 2.4:** Page of Point Communications showing links to lists of pages which rated
In the case of Point, each visited page is given a score for each criterion ranging between 0 and 40. The scores are then added to form the total score of the page. Pages are ranked using the arrived total.

**Problems with Subjective Metrics**

A discussion of whether Point’s scheme of ranking pages is valid is difficult and not particularly enlightening. More interesting would be a discussion of the advantages and disadvantages of such a metric for determining a page’s popularity.

In terms of advantages, any metric which relies on human judgement to predict a page’s popularity has distinct advantages over automated mechanisms like those described previously. These advantages stem from the fact that subjective metrics reflect what users prefer to see in pages. A judge is essentially an experienced web surfer. If he predicts that a page will be popular, that page will likely become so because his judgement mirrors that of other human users who surf the web. This is the same reason why stock markets, for example, sink or rise
depending on the opinions of stock brokers and brokerages even though computer stock pickers have become increasingly sophisticated - by using human judges to make decisions regarding a product’s demand when that product’s popularity is heavily affected by human perception, issues too subtle to an automated mechanism can be detected and incorporated into a prediction of a page’s popularity.

On the other hand, computers are useful because their perceptions are based on consistent criteria and they are fast, whereas humans fall short on both points. A person may find a page interesting one day, not so interesting the next. Humans are also slow and are more prone to make mistakes in random fashion, whereas computers are fast and, if they make mistakes, commit them consistently; once the source of their errors is discovered and fixed, they cease to commit those same mistakes again.

These two issues cast doubt of the usefulness of subjective metrics for predicting a page’s popularity simply because a judge is unlikely to have seen a good portion of the web and is likely to have an opinion which wavers often.

2.2 The Need for Normative Guidelines for Making Automated Comparisons

So far, this chapter has listed a number of shortcomings with different mechanisms for determining a web page’s usage. For each shortcoming, it is conceivable that improvements to the data analysis mechanism (hit counts, link counts, etc.) can be made to minimize the deficiency’s effect. However, any such improvement does not combat the fact that the set of web pages which need to be analyzed is too large and overly diverse.
If instead of improving the mechanism, one could reduce the set of web pages over which the mechanism is to be performed, the analysis technique, in turn, would need not to be so general.

For example, consider the problem with analyzing hit count data due to the lack of differentiation between hits from a single site and hits from multiple sites. A common analysis technique which can be employed to improve the hit count mechanism would be to clump hits which are close to each other and which originate from the same source as forming one large access or hit, the argument being that hits originating from the same place at approximately the same time are probably from the same individual. However, while such a solution seems to work, it may fail to consider the difference between two hits from the same place over a span of two minutes and one hundred hits from the same place over the same time span. One would suspect that the 100 hits add greater value to the overall usage of the page in question yet the analysis technique mentioned before would consider both hit profiles to be identical.

On the other hand, if the only pages to which the page in question would be compared are pages of the same usage characteristics, clumping hits together will not have as severe an effect on the evaluation since all pages of interest will suffer from clumping to a similar degree.

The key idea is that it is possible to create an automated (i.e., fast) scheme which mimics the way humans judge, and categorizes pages simply by codifying how humans perceive a page’s purpose. Humans are good at perceiving purpose. Furthermore, their perception of purpose is often dictated by the structure of the pages they see. An automated scheme which reduces a web page into its HTML structural skeleton can use statistical tools to predict the purpose for which that page was created. This prediction, while not completely fail-safe, will be efficient enough so that for most cases it determines a web page’s purpose correctly. More-
over, because the scheme breaks down pages more precisely and its judgement system does not waver, the decisions it makes will not vary over time and will be more consistent.
Chapter 3

A Classification Strategy

3.1 A New Method for Classification

The keys to a usable categorization scheme are speed and pattern recognition. Speed deals with the amount of time it takes to examine and categorize a page. Since we can assume that a scheme which has no human component can be computed as quickly as desired, a scheme's ability to be automated is crucial. Pattern recognition deals with the ability of the scheme to elicit function or purpose from the web page being classified.

As mentioned previously, the common popularity metrics suffered problems either because evaluating the metric on a page took too much time (subjective metrics) or the metric failed to take functional or purpose information into account (hit count and link count). In each case, a metric chose speed over pattern recognition or vice versa. Moreover, since humans tend to change their minds over time, subjective metrics can also be said to have sacrificed consistency for perception. In this section, it will be shown that a classification scheme which compromises between the two properties is achievable and can lead to a better understanding of what makes pages popular.

In this chapter, background and initial work on a strategy for categorizing web pages is presented in three parts. First, a discussion of background material on genres and function types will be presented. These topics will form the framework for the proposed categorization scheme. Next, previous work which applied genre classification on categorizing web pages will be summarized. Lastly, the systematic classification methodology used in [3] will be combined with the concept of function types to form a classification scheme which can easily be automated.
3.2 The Concept of a Genre

A genre is a recognized and accepted form of communication. Instances of a particular genre share aspects of form, content, or purpose. Genres are thus the common structures by which communication in a particular medium is carried out.

Recently, Yates and Orlikowski proposed using genres as a basis for studying communication within organizations. They defined genres as “a distinctive type of communicative action, characterized by a socially recognized communicative purpose and common aspects of form.” In other words, genres arise due to socially recognized needs for communication. As individuals communicate similar social motives, topics and themes, they tend to do so using methods which are similar in form and structure.

3.3 Previous Work

Much work has been done previously on genres and the classification of web pages. “Genre Repertoire: The Structuring of Communicative Practices in Organizations” by Yates and Orlikowski [7] proposed the notions of genre and genre repertoire as “analytic tools for investigating the structuring of communicative practices within a community.” The prime example used was a case study of electronic mail (email) communication among a group of computer language designers defined loosely as the Common Lisp (CL) group. Through the early 80s, the CL group engaged in complex negotiations to produce a standardized version of the artificial intelligence language LISP which they dubbed Common LISP. And because these researchers were located at universities and company sites dispersed geographically throughout the US, email became the primary means by which they interacted.
In that case study, Yates and Orlikowski showed that several genres arose in the email communication amongst the CL group. The primary genres were the memo, the proposal and the dialogue. Also, a genre system (the ballot) was used on a regular basis.

Recently, the concept of genres has been applied to web pages. In [3], Crowston and Williams classified a set of randomly chosen web pages, proposing the existence of at least 34 web page genres, including the homepage, the FAQ, the directory and the filmography. Each genre represented a distinct class of web pages whose form and purpose were unique from those of different genres.

3.4 Classification Strategies and Popularity

This project takes the work of Yates and Orlikowski, and of Crowston and Williams, and applies it to a real world problem: defining a basis for comparing web pages. More precisely, the goal of this project is to arrive at a routine means of classifying web pages, and to show that such a normative classification allows a more reasonable comparison of hit counts and link counts of web pages.

Previously, it was argued that there was a distinct need for a normative guideline for making comparisons of web pages. In light of [3], one might argue that the web genres of Crowston and Williams would do. Such an argument would hold true if classifying web pages into genres were easy.

However, genre level classification requires manual coding of webpages, which is slow, tedious, and prone to errors and bias. Moreover, a human's judgment system varies from day to day. What might appear to be a list one day might end up being categorized as an FAQ the next. And while admittedly human judgement is superior to any automated classification scheme in terms of how much information it uses in making its classifica-
tion, when one considers the current size and rapid growth of the web, and realizes how crucial automation is in any web classification scheme, genre level classification's dependence on human involvement makes it unsuitable as a basis for classification.

In light of this difficulty, a categorization metric which is less fine-grained but easily automated becomes the best option for two reasons: it is fast and, if it is biased, it is biased consistently. Moreover, as long as the metric does not change, the categorizations made by the automated system will be time independent; a page will be categorized the same regardless of whether it is the first page categorized or the last.

3.5 The Concept of a Function Type

A function type may be considered a meta-genre of communication. It encompasses a set of purposes and related form features. The metrics which define what function types a class of communication is subdivided into is dependent on what features one seeks to examine. For example, one may choose to partition email according to the number of recipients of that email (one or many), if such a classification were to prove useful. In the case of web pages, we choose to delineate web function types according to the flow of information between user and web author.

The classification scheme adapted here categorizes pages based on the apparent flow of information between page user and page author. The reason for choosing this basis is because flow of information can easily be inferred from the structure of the web page; if a page is fully text, then flow is from page to user; if it is fully fill-out forms, then flow is in the other direction; and so on. Since flow of information is a characteristic of purpose, determining the flow is equivalent to inferring (part of) the purpose the author had for the page's existence.
The categorization scheme proposes the existence of 5 unique function types, each with its own corresponding characteristics. These function types are described in table 3.1.

<table>
<thead>
<tr>
<th>Function Type</th>
<th>Definition</th>
<th>Characteristics</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>transactional</td>
<td>users access these pages to provide information for a service or transaction</td>
<td>substantial use of fill-out forms, buttons, editable text regions, etc.</td>
<td>surveys, solicitation forms, signup sheets</td>
</tr>
<tr>
<td>informational</td>
<td>users access these pages to obtain information</td>
<td>low link counts and high screen densities.</td>
<td>FAQs, newsletters, books</td>
</tr>
<tr>
<td>interactional</td>
<td>users access these pages repeatedly to seek new query results.</td>
<td>balance between fillout forms, text and links.</td>
<td>muds, puzzle pages, databases</td>
</tr>
<tr>
<td>referential</td>
<td>users access these pages as jump points to other pages</td>
<td>high link counts, use of lists of links, high percentage of linking text</td>
<td>maps, lists</td>
</tr>
<tr>
<td>hybrids</td>
<td>mix if the above types</td>
<td>multiple sections, usage of &lt;chr&gt; and &lt;div&gt; anchors</td>
<td>vitae, resume, homepages</td>
</tr>
</tbody>
</table>

Table 3.1: Function Types and Their Specifications

Several things of note need to be mentioned at this point and will be more fully developed in the next chapter. First, the boundary between any two groups is somewhat arbitrary. Because HTML started out as an information presentation language and has only been around for 8 years, the newer HTML features which allow for more complex page-to-reader interactions are underutilized. Thus, one would expect a substantial minority of pages to fall under the second group, with reference pages being the next most popular. However, as the web matures and newer versions of the HTML language are formalized, for example, interactional pages should increase in number.

Second, the WWW and HTML are volatile communication media. As HTML matures under the auspices of the World Wide Web Consortium and Web vendors like Oracle, Netscape and Microsoft, new genres will arise which may not strictly fall under any of the first four function types. Moreover, even today, HTML authors are encouraged to mix and
match function types in pursuit of the eye-catching web page. Therefore, there is a need to keep the classification scheme open. The hybrid function type was added in recognition of this volatile creativity and gives the current classification scheme some room to grow.

Lastly, because the classification scheme classifies a page solely on the relative abundance of different HTML form features on that page, it is a scheme which can easily be automated. To automate the classification, a sample of manually classified pages must be sorted into the five categories and the structures of the pages examined. The structures can then be condensed to form HTML profiles for each category, a profile being a set of tolerances for each HTML form feature. Classification of new pages would then involve fitting pages to these profiles.

This fact gives this classification strategy a distinct advantage over classification strategies which require human involvement such as that employed in [3] and in Point Communications’ Top 5% ranking system.
Chapter 4

Methodology

4.1 Reason and Motivation

If a categorization scheme were to prove useful, one would expect pages which were similarly classified to exhibit structural similarities. Thus, to demonstrate the power of this functional categorization scheme, the structural and functional makeup of two sets of pages, one popular and one random, were analyzed before and after they were categorized by function type.

4.2 Overview of Methodology

The above task was split down into three distinct stages, collection, categorization and analysis.

In the collection stage, both the pages in Lycos 250 and 250 random pages were collected. Gathering the Lycos 250 set was accomplished by manually fetching the HTML source for each page listed by Lycos in its top 250 list and storing it into a file. On the other hand, collecting the 250 random pages was slightly trickier. To obtain this set, 8000 HTML addresses were collected from the WWW over a period of six weeks. Collection of the 8000 base pages was accomplished by constructing a web spider and allowing the spider to forage about the Internet starting from pages generated at random by clicking on the “Anything Goes’ link under the Surprises category of the Alta-Vista search engine put up by Digital Equipment Corporation. Each run involved allowing the spider to wander the Internet until it had found 500 pages which were worth cataloguing (i.e., an actual page was retrieved from the site as opposed to an error message). During hours when internet
usage in the United States was at its peak, fetching pages often took longer. Oftentimes, the pages would arrive incomplete or would not arrive at all. Thus, to ensure that a maximum number of pages was fetched correctly and to minimize the amount of time spent waiting for fetches to complete, collection of web pages was done in the early morning between 3 and 5am EST.

After the 8000-page HTML address set had been retrieved, a random sample of 250 pages corresponding to an equivalent number of addresses in the address set was collected. By choosing randomly from a large pool of http addresses, I attempted to make the pages selected be sufficiently random and thus representative of what the WWW had to offer. These pages (i.e., the HTML source code of these pages) were downloaded into separate files in a directory (i.e., each file corresponding to a different page). In each case, the HTML address was the filename. This choice facilitated later processing, when the correspondence between page and html address became crucial.

The categorization stage involved classifying each page into one of the five function type categories. A page was categorized by applying each of the five function-type profiles to the page. If the page fit a particular profile, this would imply that its structural makeup fell within the tolerances for the corresponding function type, and should be thus classified.

Profiles for each category were developed by manually categorizing a small sample of web pages and deriving structural heuristics for each function type. For example, a page whose HTML code size was large and approximately equivalent to its text size would probably be an informational page. On the other hand, a page which only had a list of links would clearly be a reference page. Automated techniques derivable from perceptron theory [17] could have been used here but in light of the time restrictions of the project, they were not.
Because categorization had to be done automatically, most pages which did not follow
the HTML 3.0 grammar were discarded. The pages which failed the specification but were
still kept were those which violated the grammar in trivial ways, such as the use of the
<header> label as opposed to <head> and failure to wrap the head of the HTML file
around <head> and </head> labels. The remaining pages which failed the HTML 3.0
specification were removed from the eventual sample set because while they may look
well-structured on a version of Netscape or some other browser, it is very likely that the
page will not appear properly in most other browsers and may not satisfy the expectations
of both the web author and the client perusing the page.

It should be noted that a page which had correct form would be presented correctly by
all browsers. And while it may be argued that Netscape’s dominance in the browser mar-
ket (and its continued violation of HTML specifications) may mean that authors need only
ensure that their pages look right in all versions of Netscape’s browser, new browsers
released by companies like Oracle, Microsoft, Sun, Next and Spyglass will not present
‘Netscape optimized’ pages correctly. As more established internet companies weigh in
with their own browsers, most users will surf the web using browsers which cannot
present these pages correctly. Thus, following HTML specification is in itself a good mea-
sure of how popular a page would become, as indicated further by the high percentage of
popular pages following the HTML grammar.

Finally, additional random pages were chosen to equalize the number of random popu-
lar pages analyzed.

The analysis stage involved contrasting the structural and functional makeup of the
popular pages against those of the randomly chosen pages, before and after categorization
was performed.
Analysis of web pages before they were categorized involved deconstructing the pages in each popularity group along the grammar rules of HTML 3.0 and collecting statistics regarding the relative abundance of particular HTML features in each page. By comparing the statistics of the Lycos 250 pages against that of the set of 250 random pages and developing t-tests for these statistics, the correlation between the count of a particular feature and a page's popularity could be determined.

Analysis of web pages after categorization was done in a similar fashion. Pages falling into the same category and popularity group were deconstructed as before and their features catalogued. Statistics of the Lycos 250 pages falling in some category were compared to statistics of the random set falling in the same category.

4.3 Software Design

Software was written to accomplish the automated portions of the project, including selecting the random pages from the WWW, performing function type categorization, and analyzing the structural makeup of the web pages after each round of categorization.

Three software modules were used to select and categorize the 250 random pages automatically: a web page address engine called a net spider or net wanderer, a data extraction machine, and a filter set. Spanning these three modules was a database system through which the information generated by the net wanderer could be shared among the three modules.

The net wanderer was used to collect http addresses of web pages. These addresses were then stored in the database for use by the next module. Once a sufficiently large number of web page addresses had been collected, the wanderer would terminate its operation.
The data extraction module selected random addresses from the database and fetched the associated pages (i.e., the source code of the html pages). These pages were stored in a directory for later analysis.

The third module, called the filter set, was used both to categorize and analyze the web pages. This module was capable of breaking pages down into their constituent primitives (i.e., HTML forms), cataloguing the counts of each primitive, and from that data, categorizing the pages along the metrics discussed in the previous section.

**Web Spider: Details**

The web spider collected HTTP addresses in the following fashion. Starting at an initial web page, it would first strip a page of all the HTTP addresses the page contained. Each address would be inserted into an address queue if the spider deemed the address interesting. After it had considered all the page's addresses, it would discard the page, pull the first address from the queue and fetch the corresponding page, doing to that page what it had done to the previous. The spider would continue in like fashion as long as the queue was not empty and the number of pages it was tasked to collect had not been reached. Because fetching a page corresponded to sending a request over the internet to an unknown machine, the spider was designed to stop its fetch once a time-out period had been reached during which time no response from the target machine had been received.

The spider regarded an http address as interesting depending on two criteria: one, if it had never seen the address and two, if the address was sufficiently different from the address of the page it was currently examining. The first rule prevented the spider from examining a page multiple times while the second rule ensured that it was sufficiently mobile (i.e., it did not spend all of its time wandering in a single site but explored pages
outside that site). Pages residing close together tended to resemble one another since the authors of these pages often copied from each another. By increasing the web coverage of the spider, I was attempting to assure that the pages eventually selected at random would be more representative of the kinds of pages on the WWW.

**Figure 4.1:** Graphic representation of spider traversing html links

![Graphic representation of spider traversing html links](image)

The spider was also built so that it followed a set of spider etiquette rules adopted by the guild of web spider makers. These rules, as specified in [5], defined how robots were to behave if they were to be left unmonitored on the internet and also gave provisions for web site owners to provide maps to these robots which would inform them what pages on their sites were off-limits to spiders. The spider was written to satisfy most of the rules. However, because of an early design decision made prior to my discovery of these principles, the rule requiring robots to identify themselves to web sites could not be followed without a complete overhaul of the web spider exploration routines. Because of this failure, the spider was never allowed to explore the internet unmonitored.

**Collector: Details**

The collector module functioned in a simple way. Given a database of http addresses and a number of pages to fetch, the collector module would query the database for a ran-
dom http address. Given that http address, the collector would download the file located at that http address and save it in some file. It would continue to do so until it had collected the requisite number of pages it was asked to fetch.

**Filter Set: Details**

The filter set is a package of procedures which extract from a web page all instances of a particular HTML feature (e.g., table, number of characters, etc.). Written in Perl, these routines formed the basis for a parser of HTML pages based on the grammar rules specified by HTML 3.0[2].

These procedures were used for two purposes. First, they were used to implement the categorization scheme, as well as determine whether a page satisfied the HTML 3.0 specifications. A page which could not be parsed correctly failed the specifications and by implication could not be categorized by this software. Second, the routines were used in performing the statistical analysis of the web pages once the categorization of the pages had been completed. Each filter routine could retrieve mentions of a particular HTML form feature. Thus, by counting the numbers returned by the filters for a variety of HTML form features, the filter could construct a basic structural profile of the pages in a single category and popularity class.

**Problematic Issues of the Software Implementation**

The software described above was augmented with several add-ons in light of technical difficulties encountered later in the experiment.

First, the difficulty of categorization meant that not everything could be automated. There was no guarantee that the categorization would always work since no effort was
made to ensure that the heuristics covered the space of possible web pages. Thus, the possibility of having to manually categorize a web page had to be considered. Also, as mentioned before, certain pages with trivial HTML errors were altered after they had failed initial categorization. These pages were then fed back into the categorization scheme for a second run after changes had been made by hand.

Second, the need for a random set of web pages meant that the 8000 http addresses had to be selected from a wide assortment of sites. Because the spider was prone to investigate pages which were very related to each other, it was made to do explorations starting from different web sites. These web sites in turn were obtained from the 'Anything Goes' random http address generator available through the Dec Alta-Vista spider.
Chapter 5

Results

5.1 On Selecting and Cleaning the Sample

All 250 pages in Lycos 250 were fetched and examined. Of these pages, 146 (55.2%) satisfied the HTML 3.0 language specification or satisfied the specifications enough so that only small alterations in the source code were needed for the pages to be usable. Common errors in source code which were considered fixable included pages which neglected to use either the head or the body anchors, and pages which used the word “header” instead of “head” for the head anchors. Common unfixable code errors included pages without titles, missing heads, or titles located in wrong places (i.e., body or outside the html code proper).

Of the 250 random pages selected, only 101 (40.4%), including pages which had minor errors, satisfied the same criteria. To allow adequate comparisons at subsequent stages of analysis, further harvesting of random web pages from the WWW was performed to boost the number of usable pages in the random set until it equaled the 146 of the Lycos 250.

5.2 Collection of Statistics

Information regarding the structural makeup of the pages was obtained using the procedures comprising the filter set and, in the case of screen size, by visual inspection (see table 5.1 for statistics). The properties of an HTML document’s structure deemed more revealing of the document’s usage were collected, as follows:

* body code size - the number of characters of the HTML code forming the document’s
body (i.e., excluding that forming the head)

- link count - the number of http links in the document
- image graphics count - the number of gif, jpgs and other pictures in the document
- form count - the number of fillout forms in the document
- table count - the number of HTML tables in the document
- list count - the number of lists in the document
- screen count - the number of screens which Netscape took to display the entire document. Default settings for text size and fixed font, and encoding were chosen. Moreover, the Netscape cache was set at 10megabytes to improve fetching speed. Count for this statistic was done manually.
- text size - the number of characters displayed by lynx, a text-based browser, when viewing the HTML document

<table>
<thead>
<tr>
<th></th>
<th>Lycos</th>
<th>Random</th>
<th>abs(t)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Body Code Size</td>
<td>6022.38(8656.61)</td>
<td>27371.8(122271.7)</td>
<td>25.38*</td>
</tr>
<tr>
<td>Images</td>
<td>7.07(7.92)</td>
<td>5.39(9.70)</td>
<td>1.94*</td>
</tr>
<tr>
<td>Form Count</td>
<td>0.205(0.77)</td>
<td>0.21(1.66)</td>
<td>N/A</td>
</tr>
<tr>
<td>Table Count</td>
<td>0.404(1.03)</td>
<td>0.46(1.98)</td>
<td>N/A</td>
</tr>
<tr>
<td>List Count</td>
<td>1.92(3.97)</td>
<td>4.39(24.50)</td>
<td>1.46</td>
</tr>
<tr>
<td>No. of Links</td>
<td>45.29(86.20)</td>
<td>39.03(94.99)</td>
<td>0.75</td>
</tr>
<tr>
<td>Screen Count</td>
<td>2.89(3.55)</td>
<td>11.76(45.73)</td>
<td>2.82**</td>
</tr>
<tr>
<td>Text Size</td>
<td>1974.08(921.03)</td>
<td>11719.90(22865.3)</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table 5.1: Comparison of Means of Form Features Between Lycos and the Random Set. *p<0.05; **p<0.01

In addition, a number of other features were derived from the above (See table 5.2). These were obtained by dividing the number of links, the text size, number of lists, and number of images respectively by screen count. The resulting numbers reflected a per-screen average of each HTML 3.0 feature for the corresponding document.

The form features for the 146 pages from Lycos and the random set were compared by using the t-test for the difference of means. The features which proved to be statistically significant (p < 0.05 or less) were body code size, images, screen count, link density and
image density. T values were not available for several of the statistics since the corresponding features were almost never used in the sample obtained.

<table>
<thead>
<tr>
<th></th>
<th>Lycos</th>
<th>Random</th>
<th>abs(t)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Link Density</td>
<td>25.67(46.70)</td>
<td>6.65(9.47)</td>
<td>3.41**</td>
</tr>
<tr>
<td>Text Density</td>
<td>683.07(3154.78)</td>
<td>1171.48(715.34)</td>
<td>1.29</td>
</tr>
<tr>
<td>List Density</td>
<td>0.66(1.35)</td>
<td>0.37(0.98)</td>
<td>1.48</td>
</tr>
<tr>
<td>Image Density</td>
<td>2.45(4.11)</td>
<td>0.46(3.39)</td>
<td>3.19**</td>
</tr>
</tbody>
</table>

Table 5.2: Normalized Features Derived from Features of Referential Lycos and the Random Set Summarized in Previous Table. *p < 0.05, **p < 0.01

5.3 Comparison of Pages

Lycos vs. Random: (Before Categorization)

A number of differences could be observed between Lycos and the random set simply by leafing through a selection of the pages in either group (see tables 5.1 and 5.2 for a summary of the statistics). Lycos pages were generally short and concise, as indicated by text size (< 1974.08 characters per page) and screen count (< 2.89 screens on average). These pages exploited the hypertext characteristics of HTML to the full, often presenting in the top part of the page a gross summary of what the rest of the page and the pages associated with it were about. The statistics bore this fact out, with the average screen count of Lycos pages around one quarter the count for the random set (average of 2.89 screens for the Lycos vs. 11.76 screens for the random set). Naturally, a smaller screen count implied that Lycos pages would have shorter text sizes, less body code and body text sizes than pages in the random set.
Examination of the normalized features showed that Lycos pages also had twice the average link density (14.66 links per screen for Lycos vs. 6.65 links per screen for the random set) of the random set, while surprisingly its text density stayed the same. Lycos pages had twice the number of links per page, and its image density was five times that of the random set. All other features were statistically insignificant.

**Lycos vs. Random: (After Function Type Categorization)**

Every page was categorized into one of five categories: informational, transactional, interactional, referential and hybrid (see Table 5.3). Of the 146 Lycos pages, a majority were classified as referential pages (77 out of 146 or 52.74%), with hybrid pages coming in second (35 out of 146 or 23.97%) and informational pages third (29 out of 146 or 19.86%). Of the 146 random pages, a majority were classified as informational pages (88 out of 146 or 60.27%), with referential pages coming in second (38 out of 146 or 26.03%) and hybrid third (12 out of 146 or 8.22%). The number of interactional and transactional pages in the Lycos and random sets were too small to merit any further study.

<table>
<thead>
<tr>
<th>Category</th>
<th>Lycos</th>
<th>Random</th>
</tr>
</thead>
<tbody>
<tr>
<td>Informational</td>
<td>29</td>
<td>88</td>
</tr>
<tr>
<td>Referential</td>
<td>77</td>
<td>38</td>
</tr>
<tr>
<td>Hybrid</td>
<td>35</td>
<td>12</td>
</tr>
<tr>
<td>Interactional</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Transactional</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Total(N)</td>
<td>146</td>
<td>146</td>
</tr>
</tbody>
</table>

**Table 5.3: Breakdown of the 146 pages from Lycos and from the random set**

Most Lycos pages were homepages made by and for large institutions and companies. These homepages served as overlying pages for their site, with links to all the companies
subdivisions (i.e., a referential page). On the other hand, a majority of the random pages were pages manually written by individual users. These pages fell under the informational category since they contained more descriptions of the page’s subject matter instead of links.

Using function type categorization, several other new observations came to light.

Random set pages in the referential bin differed from their Lycos counterparts in several ways (see figure 5.4). First, they were roughly twice as long in screen count as referential Lycos pages. This difference was not as big as the difference between Lycos and random pages prior to classification (1:4 ratio in screen count). Second, while Lycos and random set referential pages had approximately the same number of total images, Lycos pages had twice the number of images per screen compared to random ones. Moreover, while random set pages and Lycos pages had approximately the same average number of images, images in Lycos were often maps, graphical HTML tools which allowed users to reference pages by clicking on a portion of the picture instead of searching through lists of links. On the other hand, random set pages used pictures as decorative items and were less likely to use them as navigational/search aids. This observation was made both when manually scanning through the pages and when the source code of the HTML documents were examined. Lastly, it was shown before that the link density of Lycos pages was twice that of the random set. However, when only referential pages were considered, the link density of Lycos pages and those of the random set turned out to be comparable (17.90 links per screen for Lycos vs. 18.35 links per screen for the random set.)
In the informational category, the screen counts of Lycos pages and those in the random set were far apart, with random set pages on average taking up 5 times the number of screens as their Lycos counterparts. Despite the disparity in screen count, Lycos pages had twice the number of images per page as random set pages, ten times the image density and more than nine times the link density. These differences seem to indicate that the use of pictures makes an HTML document much more eye-catching. Moreover, the increased number of links would indicate that authors of Lycos pages would be more likely to take
advantage of the hypertext nature of HTML to create web pages which were structurally more complex. It is possible that the complexity results in documents which are better organized. For example, one interesting web page which was not included in the sample was a PhD thesis outline by a student at the Georgia Institute of Technology. Using the hypertext capabilities of HTML, the author turned the entries in the outline into links which pointed at the corresponding chapters and subchapters in his thesis.

<table>
<thead>
<tr>
<th></th>
<th>Lycos</th>
<th>Random</th>
<th>abs(t)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Body Code Size</td>
<td>6346.72(19062.42)</td>
<td>39787.36(199358.1)</td>
<td>3.63**</td>
</tr>
<tr>
<td>Images</td>
<td>7.72(7.94)</td>
<td>3.97(3.43)</td>
<td>9.43**</td>
</tr>
<tr>
<td>Form Count</td>
<td>0.04(0.196)</td>
<td>0.01(0.41)</td>
<td>N/A</td>
</tr>
<tr>
<td>Table Count</td>
<td>0.28(0.60)</td>
<td>0.50(0.31)</td>
<td>N/A</td>
</tr>
<tr>
<td>List Count</td>
<td>1.20(2.97)</td>
<td>4.65(27.31)</td>
<td>2.73**</td>
</tr>
<tr>
<td>No. of Links</td>
<td>31.68(51.67)</td>
<td>16.65(166.91)</td>
<td>1.87</td>
</tr>
<tr>
<td>Screen Count</td>
<td>3.32(3.62)</td>
<td>16.59(71.67)</td>
<td>4.02**</td>
</tr>
<tr>
<td>Text Size</td>
<td>4655.00(7339.95)</td>
<td>16,297.07(128257.6)</td>
<td>1.97*</td>
</tr>
</tbody>
</table>

Table 5.6: Comparison of Means of Form Features Between Informational Lycos and Random Set Pages. *p < 0.05, **p < 0.01

<table>
<thead>
<tr>
<th></th>
<th>Lycos</th>
<th>Random</th>
<th>abs(t)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Link Density</td>
<td>9.48(5.65)</td>
<td>1.00(51.32)</td>
<td>3.57**</td>
</tr>
<tr>
<td>Text Density</td>
<td>1100.70(520.39)</td>
<td>982.34(1576.35)</td>
<td>1.67*</td>
</tr>
<tr>
<td>List Density</td>
<td>0.36(0.49)</td>
<td>0.28(1.39)</td>
<td>1.18</td>
</tr>
<tr>
<td>Image Density</td>
<td>2.33(3.72)</td>
<td>0.24(24.41)</td>
<td>1.84**</td>
</tr>
</tbody>
</table>

Table 5.7: Normalized Features Derived from Form Features of Informational Lycos and the Random Set Summarized in the Previous Table. *p < 0.05, *p < 0.01
Unfortunately, an insufficient number of pages were collected for each of the remaining function type categories to merit a similar comparison.
Chapter 6

Overall Remarks

6.1 Satisfaction of Result

The objective of this thesis was to devise and implement an automated way of categorizing web pages which would facilitate comparisons of web pages. To demonstrate that the categorization was useful, an analysis of the resulting data had to be performed. This analysis of collected web pages before and after function type categorization revealed a number of interesting structural properties of web pages which differentiate popular from random pages and thus may be good indicators of a web page's popularity. In particular, it was shown that while Lycos web pages in general were shorter (i.e., had smaller average screen counts) (with significance up to the 0.01 level or $p < 0.01$), had more images ($p < 0.01$), and a higher link density ($p < 0.01$) than random web pages, referential Lycos pages did not exhibit as strong a difference from its random set counterparts in those categories but differed strongly in terms of list count ($p < 0.01$ as opposed to no apparent difference between referential Lycos and random) while informational Lycos pages differed further from its random set counterparts in text density ($p < 0.05$) as well.

This analysis of the varying differences of Lycos and random web pages across differing function types, together with the fact that the categorization scheme itself was easy to automate, satisfied the thesis' objective.

6.2 Implications for Web Page Design
While this thesis does not show that structural differences cause popularity, the results may be used to provide a preliminary answer to the question “what makes a popular web page.”

In general, table 5.2 implies that the author of a web page should use screen space wisely. Popular web pages are typically eye-catching (with many pictures) and deliver what they promise in the first 3 screenfuls.

Analysis at the function type level further reveals that depending on what the page is for, certain formats work better than others. Popular referential pages, for example, tend to depend less on lists than random referential pages and to have slightly lower screen counts while having more images per screen and approximately the same number of links per screen as an average HTML page. Moreover, the images in popular referential pages are usually image maps. Therefore, these differences would indicate that someone planning to create a referential web page should not only keep the page to under three screenfuls but also use image maps as indexing tools instead of a list of links. Popular informational pages, on the other hand, have significantly more images and links, more lists per screen and yet lower screen counts. These differences would indicate that those planning to create informational web pages should use pictures to make their pages more visual. Moreover, they should use links to create text regions which refer to pages which elaborate on the ideas touched in the text.

6.3 Improvements and Extensions to the Categorization Scheme and its Dependent Algorithms

As stated before, the primary objective of the project was to arrive at a categorization scheme which could be automated and would facilitate the assessment of web page form and function features, and would suggest some indicators of popular pages. In light of this
goal, there are several ideals which were not achieved in the current implementation of the project.

First, automated categorization was done only at the function-type level. While some facility was obtained from this scheme, it might be enlightening to see if it is feasible to automate categorization at the genre level. Humans develop heuristics for categorizing web pages into genres. If it is possible to codify these heuristics and arrive at a set of 'fuzzy' rules which would encompass a majority if not all of the common heuristics used, then an automated scheme for categorizing at the genre level could be created. Examining what features are important for each genre class would be of extreme interest, since it would reveal how exactly humans enact genres.

Second, in the parsing of the web pages, some improvement could be made to the parsing scheme so that the parser returns not only a feature count of a set of HTML forms for each page, but also presents an actual structural skeleton. A structural skeleton contains much more information than a feature count and may prove to be a more flexible basis for study since it also contains details about how individual divisions of a web page are structured. At the moment, the current system makes no delineation between different sections of the body since very few web authors made use of the <dvi> anchor (i.e., section anchor) provided for by HTML 3.0. In the future, use of the section anchor may lead to structural analysis which actually creates a structural profile instead of a statistical profile as is done now.

Third, with the advent of the Web programming language Java from Sun Microsystems, a further extension of the project could be made to take this new HTML specification into account. Dealing with Java is tantamount to opening a huge can of technological worms since determining a web page's use would require an automated means of examining a program's output and display stream. However, pages which fall under the interac-
tional category should increase in number, since Java would allow the addition of new client-side and server-side ‘interactive’ capabilities to an already rich web page functionality, functionality which previously was only achievable using hard to use server-side CGI scripts.

Lastly, it might be interesting to extend the study from page examination to site examination. Such a study would bring categorization into a different level, where web pages could be examined as complete hypertext documents and not simply as two-dimensional sheets. The use of the added dimension could then be assessed and a comparison of web page hierarchies of popular and random pages could be made.

Unfortunately, certain issues need to be resolved in order for such a study to be possible. The main problem would be the difficulty of determining what exactly defines the borders of a site. A site is a congruous collection of web pages such that the whole set of pages in the site concern a specific topic. It used to be that sites and servers corresponded one to one. However, a single web server may now host more than one site. For example, many of the HP web sites reside on the same HP web servers. Furthermore, a site may be a collection of web pages strewn over a number of servers, as is the case for commercial web sites like those maintained by Dell Computer (http://www.dell.com). Lastly, a page may be a member of multiple sites, in the same way that a student’s page could be a page belonging to several sites which are separately owned by the different organizations of which the student is a participant.

Defining a site is not as simple as examining a page’s HTML address to ascertain the page’s host machine. It also requires an examination of the page’s content and purpose. With the nebulous nature of the world wide web and the continued evolution of the HTML language, it is unlikely that any defined edges for a site could be clearly specified.
Other, more minor improvements may be made to increase the efficiency of the web spider’s search algorithm, and keep the spider abreast with the evolving specifications of both HTML and robots.txt. These improvements are a necessity if the scheme introduced in this thesis is to adapt to the rapidly evolving view of HTML and the WWW.

6.4 Improvements in Code Efficiency

First of all, it should be pointed out that the choice of Perl as the base programming language for this project was borne out of time constraints. In other words, Perl became the language of choice (and the language of choice for a majority of the robots on the internet) because it was very powerful, encompassed popular features of different languages from a flexible lexical analyzer (taken from lex, awk, two popular lexical analyzer programs), to its somewhat clean procedural interface (taken from C, shell scripts), to a straightforward melding with files and file structures (taken from shell scripts) and to its support of a simplistic database.

Unfortunately, the use of Perl has led to a number of efficiency problems. In particular, the efficiency of the spider hinges critically on the efficiency of the underlying database embedded in Perl, which is actually a hash table with automatic linking to an associated file. While the size of the sample set has remained manageable, the performance of the database has not played a crucial role on the runtime of the entire sampling routine. In fact, as can be seen in the methodology and the results sections of this paper, no detailed analysis was made to sample runtime since the sample set was chosen beforehand to be small enough so that sampling could be done over a few short weeks.

In order for this project to be useful by something as large in scale as Lycos, Excite or any of the other web search engines, runtime performance needs to be evaluated. This
evaluation may point to the need for a database specialized for the needs of the automated categorization scheme.

Furthermore, the project may benefit from the adoption of a more efficient language like C or C++ as the base language for writing all of the networking and filtering routines. Such a language, together with commercial software which gives the language access to databases, would lead to software which will be much faster than that which was implemented for this project. Clearly, the speed gain is traded off in exchange for the ease of coding in Perl. However, considering the value to be achieved by having a fast web sampler and filter set, the trade-off may not be an option but a necessity.

6.5 The Need for Complete Coding Automation

The current implementation resembles a toolbox more than a true package. In other words, little effort was made to ensure that different modules worked together to produce an output which was intelligible to a naive user. Instead, obtaining the statistical data from the data input involved distinct steps performed individually using distinct software modules. Sampling on the web was done using the spider. The results of the spider were passed on to the filter set and to the categorization scheme. The categorization scheme and filter sets returned the statistical makeup of each category and a ready-made spreadsheet was used to collate and derive further statistics from the data generated by the filter sets.

Moreover, the project was developed and carried out in parts, the sampling of the web was done in parallel with code writing for the categorization scheme. Consequently, no effort was made to create a single program which would sample the web, categorize the web pages sampled and present a statistical analysis of the categories obtained. Again, the
result was lost efficiency and the need for close human involvement during the collection and categorization steps.

6.6 The Need for HTML Stability

There is a strong need for stabilization of the HTML specification in order for this categorization scheme to become workable. At the moment, HTML defines an extremely volatile space of hypertexts. In order for the categorization scheme to work, a majority of the pages used must stick to one stable version of HTML (of some higher version number). Otherwise, comparing pages would require the categorization scheme to restrict itself to pages which use the oldest version of HTML still in common use. In turn, this requirement would mean that the scheme would be blind to HTML forms available only in more advanced versions of the markup language. And while categorization would still be correct vis-a-vis the heuristics used, they would not be very useful for the users at large who rely on the categorization scheme to set for them search bins for restricting their queries.

6.7 Restrictions of Structure-based Classification Schemes

It may be argued that restricting a search to a structure-based category artificially restricts the search along structural parameters. Such a restriction is artificial because searches are done along purpose-based lines. Crowston and Williams [3], for example, argue that distinguishing web media by physical form is not very useful.

While such an argument may be valid, it should be noted that there is a very strong association between structure and purpose. While the correspondence is not one to one, it
certainly exists. Since purpose-based categories are subjective and hard to formalize, structure-based classification is a good first cut at differentiating web pages.

Moreover, it should be pointed out that this project does not restrict the categorization scheme used. Instead the thesis' statement is that SOME classification scheme should be used to preclassify web pages into exclusive classes which can then be composed and, perhaps in the future, even searched more efficiently by a search engine. If it is possible to formalize and automate a purpose-based classification scheme, then such a scheme can supplant the structure-based method used here.
References


Appendix A

Code

# Code for get_picture.pl
# input: an html 3.0 file
# output: a list of the images resident in the html file
#!/usr/athena/bin/perl
sub get_pictures {
    local($htmlfile)=$_[0];
    local(@htmlfile_q);
    # See if the document has a comment element
    while($htmlfile =~ /(<!--[^>]*-->)/i) {$htmlfile = $';
     push(@htmlfile_q, $1);
     print("$1\n");
    } return(@htmlfile_q); }
$count = $#ARGV; $index = 0;
@picture_q;
# for all args
while($index <= $count){
    $thefilename = $ARGV[$index++];
    open(THEFILE, $thefilename) || die "Error: $!
";
    $document=join( "<THEFILE>");
    @picture_q = &get_pictures($document);
    print("Number of pictures: ", $#picture_q + 1 , ",\n");
}
# Code for get_sample.pl
#!/usr/athena/bin/perl
# randomize srand(time$$);
# check number of args
if($#ARGV != 0){
    # exactly one arg print("Usage: get_sample.pl <positive num>\n"); exit(1);
}
$num = $ARGV[0];
# open database if possible. Complain if not.
dbopen(%visited, "/mit/vbchen/web_pop/pages/database/visited2", 0666) || die "Can't open visited database: $!
";
@keys= keys %visited;
while($num > 0){
    $cur_http = splice(@keys, rand @keys, 1);
    $num--;
    print("Visiting: $cur_http\n");
    $FILENAME = $cur_http;
    $FILENAME =~ s/http//;
    $FILENAME =~ s/AW*//;
    $FILENAME =~ s/#/;g;
    #open(THEFILE, ">$FILENAME");
    #print(THEFILE "Visiting:\n $cur_http\nHTML CODE:\n");
    #close(THEFILE); system("lynx -source $cur_http >> $FILENAME");
}
dbcnclose(% visited);

#!/usr/athena/bin/perl
# Copyright (c) 1995-6 Beethoven Cheng. All rights reserved.
# --------------------------------------------------------------
# Procedure takes an html file and classifies the page according to rules
# stated in the function type page.
# Usage: classify <html page>
# Requires: Web page <html page> must satisfy HTML 3.0 specifications
# a comment is bounded by a '<!' and a '-->'
# <! this is a comment -->
# takes in: an html filestring
# returns: an array of all the comments with the comment anchors removed
sub get_comment {
    local($comment)=$_[0];
    local(@comment_q);
    # See if the document has a comment element
    while($comment =~ /<!/) {
        $comment = $';
        $comment =~ /-->/;
        push(@comment, $');
    }
sub get_head {
    local($head)=$_[0];
    if($head =-/<head[^>]*>/i) {
        $head = $';
        $head =-/<\/head>/i;
        $head = $';
    }
    # If not send back "No Head"
    else {
        $head="No Head";
    }
    return($head);
}

sub get_body {
    local($document)=$_[0];
    local($body);
    # See if the document has a body element
    if($document =- <body[^>]*>/i) { $body = $';
    }
$body =~ /<body>/i;

$body = "$"; }

# If not send back "No body"
else {
$body = "No body";
}

return($body);
}

# a title is bounded by a '<title>' and a '</title>'
# takes in: an html headstring
# returns: a string containing the contents of the title
# Returns "No title" if no title was detected.
sub get_title {
    local($head)=$_[0];
    local($title);

    # See if the document has a title element
    if($head =~ /<title[^>]*>/i) {
        $title = "$";
        $title =~ /<title>/i;
        $title = "$";
    }

    # If not send back "No Title"
    else {
        $title="No Title";
    }

    return($title);
}

# a paragraph is bounded on the left by a '<p>'
# takes in: an html bodystring
# returns: an array containing all the paragraphs in a bodystring
sub get_paragraph {
    local($doc)=$_[0];
    local(@paragraph=q);
    local($attributes);
    while($doc =~ /<p>/i) {
        $doc = "$";
        $doc =~ />/i;
        $attributes = "$";
        
        ...
$doc = "$";
$doc =~ /<p>/i;
push(@paragraph_q, "$");
}
return(@paragraph_q);
)

# a picture is bounded by a `<IMG` and a `>`
# takes in: an html bodystring
# returns: an array containing the pictures in the input
sub get_pictures {
    local($bodystring)=$_[0];
    local(@bodystring_q);
    while($bodystring =~ /(<img[^>]*>)/i) {
        $bodystring =~ /$/;
push(@bodystring_q, $1);
    }
    return(@bodystring_q);
    }

# an address is bounded by a `<address>` and a `</address>`
# takes in: an html bodystring
# returns: a queue containing all the addresses in the html bodystring
sub get_address {
    local($addressfile)=$_[0];
    local(@addressfile_q);
    while($addressfile =~ /(</address[^>]*>/)/i) {
        $addressfile =~ /$/;
push(@addressfile_q, $1);
    }
    return(@addressfile_q);
    }

# a table is bounded by a `<table>` and a `</table>`
# takes in: an html bodystring
# returns: a queue containing all the tables in the html bodystring
sub get_table {
    local($lines) = $_[0];
    local($attributes);
    local($contents);
    local(@tables);
    while($lines =~ /<TABLE>/i){

$'$ -= /</TAB}$/;
$attributes =$'$;
$'$ -= /</TABLE>$'/;
$contents =$'$;
push(@tables, "<TABLE $attributes> $contents </TABLE>");
$lines =$'$;
}
return(@tables);
}

# a fillout form is bounded by a '<form>' and a '</form>' # takes in: an html bodystring # returns: a queue containing all the fillout forms in the html bodystring
sub get_filloutform {
  local($filloutformfile)=$_[0]; local(@filloutformfile_q); local($attributes); local($contents);
  while($filloutformfile -= /<FORM>/i) {$filloutformfile =$'; $filloutformfile -= />/; $contents =$'; $attributes =$';
    $filloutformfile -= /</FORM>/i; $contents = $;
push(@filloutformfile_q, "<FORM $attributes> $contents </FORM>" ); $filloutformfile =$'; }
  return(@filloutformfile_q);
}

# a list is bounded by a '<dl>' and a '</dl>' or a '<ul>' and a '</ul>' or a '<ol>' and a '</ol>' (definition list, unordered list and ordered list)
# # takes in: an html bodystring # returns: a queue containing all the lists in the html bodystring
sub get_list {  local($orig_lines) = $_[0]; local(@lists); local($attributes); local($contents);
  # look for unordered lists
  while($lines -= /<UL>/i) {$' -= /</UL>/i; $attributes =$';
    $' -= /</UL>/i; $contents =$';
  }
  # look for ordered lists
  while($lines -= /<OL>/i) {$' -= /</OL>/i; $attributes =$';
    $' -= /</OL>/i; $contents =$';
  }
  # look for definition lists
  while($lines -= /<DL>/i) {$' -= /</DL>/i; $attributes =$';
    $' -= /</DL>/i; $contents =$';
  }
  return(@lists);
}

sub get_links {
  local($lines) = $_[0];
  local($attributes);
  local($contents);
  local(@links);
  # look for definition links
  while($lines -= /<A>/i) {
    $' -= /</A>/i;
    $attributes =$';
  }
}

65
$' =~ /<A>$attributes>$contents</A>/;
(contents = $';
push(@links, "<A>$attributes>$contents</A>");
$lines = $';
}
return(@links);
)
sub rm_head {
# Remove all the new line characters so the string
# is easier to search
#$document =~ s/\n/ /g;
local($head)=$document;
local($return_string)="";
# See if the document has a head element
while($head =~ /<head>/i) {
$return_string=$return_string.$head;
$head = $';
$head =~ /</head>/i;
$head = $';
}
$return_string=$return_string.$head;
return($return_string);
}
sub rm_comment {
# Remove all the new line characters so the string
# is easier to search
#$document =~ s/\n/ /g;
local($comment)=$document;
local($return_string)="";
# See if the document has a comment element
while($comment =~ /<!/i) {
$return_string=$return_string.$comment;
$comment = $';
$comment =~ /</i;
$comment = $';
}
$return_string=$return_string.$comment;
return($return_string);
sub rm_anchor {  # Remove all the new line characters so the string  # is easier to search  #$document =~ s/\n/\r/g;  local($anchor)=$document;  local($return_string)="";  # See if the document has a anchor element  while($anchor =~ /<a/i) {  $return_string=$return_string.$anchor;  $anchor =~ /</a>/i;  $anchor ="";  }  $return_string=$return_string.$anchor;  return($return_string);  }

sub rm_list {  # Remove all the new line characters so the string  # is easier to search  #$document =~ s/\n/\r/g;  local($list)=$document;  local($return_string)="";  # See if the document has a list element  while($list =~ /<DL/i) {  $return_string=$return_string.$list;  $list =~ /</DL>/i;  $list ="";  }  $return_string=$return_string.$list;  $list = $return_string;  $return_string="";  while($list =~ /<OL/i) {  $return_string=$return_string.$list;  $list =~ /</OL>/i;  $list ="";  }  $list = $return_string;  $return_string="";  }
$list = "\n";

$return_string = $return_string.$list;
$list = $return_string;
$return_string = "\n";
while($list =~ /<UL>/) {
    $return_string = $return_string.$list;
    $list = $return_string;
    $list =~ /</UL>/s;
    $list = $return_string;
}
$return_string = $return_string.$list;
return($return_string);

$count = $#ARGV;
$index = 0;
# print headings for table <TITLEBAR>
print("HTML Address, Title, Body Code Size, Body Size, Images, Filloutforms, Tables, Lists, Links\n");
# for all args
while($index <= $count){
    $thefilename = $ARGV[$index++];
    open(THEFILE, $thefilename) || die "Error: $!\n";
    $document = join("", <THEFILE>);
    #$document =~ s/</img>/g;
    #print("Page summary for: ", $thefilename, "\n");
    @comment_q = &get_comment($document);
    #print("Number of Comments: ", (#comment_q + 1), "\n");
    $head = &get_head($document);
    #print("Head size in char: ", length($head), "\n");
    $title = &get_title($head);
    #print("Title: ", $title, "\n");
    $body = &get_body($document);
    #print("Body size in char: ", length($body), "\n");
    @picture_q = &get_pictures($body);
    #print("Number of pictures: ", $#picture_q + 1, "\n");
    @address_q = &get_address($body);
    #print("No. of addresses: ", $#address_q + 1, "\n");
    @$form_q = &get_filloutform($body);

#!/usr/athena/bin/perl

sub get_link {  
  local($lines) = $_.[0];  
  local($attributes);  
  local($contents);  
  local(@links);  
  # look for definition links  
  while($lines =~ /<A/i){  
    $' = />/i;  
    $attributes = $';  
    $' = /></i;  
    $contents = $';  
    $lines = $';  
    $attributes =~ /HREF="\([^"\]\)+"/i;  
    #print($attributes);  
    push(@links, $1);  
  }  
  return(@links);  
}

if($#ARGV != 0){

# print("No. of fillout forms: ", $#filloutform_q + 1, ", "n");
@tables_q = &get_table($body);
# print("No. of tables: ", $#tables_q + 1, ", "n");
@lists = &get_list($body);
# kludge but must correspond with titlebar printout above <TITLEBAR>
@links = &get_link($body);
open(THEFILE2, $thefilename) || die "Error: $!
";
$thefilename =~ s/#N/g;
$document = join(",", <THEFILE2>);
#$document = s/\'/u/g;
$document = &rm_comment();
$document = &rm_head();
$document = &rm_anchor();
$document = &rm_list();
print(""http://$thefilename\", ", length($body), ", ", length($document), ", ", $#picture_q + 1, ", ", $#filloutform_q + 1, ", ", $#tables_q + 1, ", ", $#lists + 1, ", ", $#links + 1, ", "n");

close(THEFILE); close(THEFILE2);
print("Usage: getlinks.pl <htmlfile>");

exit(1);

)

open(THEFILE, $ARGV[0]) || die "Cannot open $ARGV[0] $!
"

$lines = join("", <THEFILE>);

$lines =~ s/\n\n/g;

@links = &get_link($lines);

while($#links > 0){

$new_link = shift(@links);

#print(shift(@links), "\n");

system("/mit/Ivbcheng/perlfilter/get_page.pl $new_link");

}

# Number of links: ", $#links + 1, "\n");

close(THEFILE);

# spider code

#!/usr/athena/bin/perl

# web robot. Does selective BFS of WWW starting at input # uses /tmp/curpage.html as repository of currently fetched web page

# Variables: set max number of pages to visit $visitcount = 5000; $next_report = 4900; # report when visit_count == $next_report

# useful subroutines

# input: two http address strings # returns true if the two strings differ enough. # currently, two strings differ enough if they are on different sites

sub differs_enough{ local($string1)=$_[0];

local($string2)=$_[1];

local($site1);

local($site2);

if($string1 =~ /http:W.*)/i{

if($string1 =~ /http:W[AVJ*)V/){

$site1 = $1;

}

else{

$site1 = $string1;

}

}

else{

die "Bad input format string 1: $string1\n";

}

if($string2 =~ /http:W.*)/i{


if($string2 =~/(http://[A-Za-z0-9\]+)/){
    $site2 = $1;
} else{
    $site2 = $string2;
} else{
    die "Bad input format string 2: $string2\n";
}
return(!$site1 eq $site2);

# open database if possible. Complain if not.
dbmopen(%visited, "visited", 0666) || die "Can't open visited database: $!
";
# check number of args
if($#ARGV != 0){
    # exactly one arg
    print("Usage: spider.pl <http address>\n"); exit(1);
}
@visit_q = ($ARGV[0]);
while(($#visit_q > -1) && ($visit_count > 0)){
    if($next_report == $visit_count){
        $next_report = $next_report - 100;
        dbmclose(%visited);
        dbmopen(%visited, "visited", 0666) || die "Can't open visited database: $!
";
        print("$visit_count left to look for. Continue? <ret>\n");
        <STDIN>;
    }
    $cur_http = shift(@visit_q);
    print("Visiting $cur_http\n");
    system("lynx -source $cur_http > /tmp/curpage.html");
    sleep(1);
    unless(open(HTMLFILE, "/tmp/curpage.html")){ sleep(1);
    }
    @lines = <HTMLFILE>;
    $lines = join("\n", @lines);
    # while($lines =~ s/<a\s*href="([A-Za-z0-9\]+)"[^>]*>/\1\n/g){
    while($lines =~ s/<a\s*href="([A-Za-z0-9\]+)"[^>]*>/\1\n/g){
        $address = $1;
    }
if($cur_http =~ /(.*)V\w*.html/){
    $context = $1;
}
else{
if($cur_http =~ /((.*)V)/){
    $context = $cur_http;
}
}

# if context doesn't exist
if(!($address =~ /(.*)http:(.*)/))
    $address = join('/', $context, $address);

if(($address !~ /.*\.(giflpjpgpl)/) & & ({$address !~ /~\.(cgi-bin)/) & &  & ({$address !~ /~\.(map.*/) & & ({$address !~ /.*\#.*/) & & (&(differs_enough($address, $cur_http)) & & ($visited{$address} != 1))){
    push(visit_q, $address);
}

close(HTMLFILE);
$visited{$curhttp} = 1;
unlink("/tmp/curlpage.html");
}
dbmclose(%visited);