A URL-String-Based Algorithm for Finding WWW Mirror Hosts

Submitted to Committee Members

Dr. Wen-Chen Hu (Chair)
Dr. Dean Hendrix
Dr. Alvin S. Lim

by

Huigang Liang

In Partial Fulfillment of the
Requirements for the
Degree of
Master of Computer Science and Software Engineering

Auburn University

August 20, 2001
Abstract

Information retrieval research on WWW revealed that more than 30% of the web consists of duplicate pages. Mirroring, the systematic replication of content over a pair of hosts, was identified as the principal cause of duplication. Finding mirror sites could help search engines and caching proxies work more efficiently. This project introduces a technique for detecting mirrored hosts from large sets of collected URLs. The syntactic analysis of URL strings is utilized to locate possible mirror host pairs. Only a small number of pages are retrieved from each host and content analysis is performed to compute their similarity scores that will be used to determine the mirroring classification. The project is composed of a web crawler, mirror-host detecting processes, and a database system. A JSWDK server that supports Servlets and JSP is used for the implementation of this project. The experimental results indicate that the proposed methods are able to detect both partial and total mirroring, and handle cases where the content is not byte-wise identical. This technique could be used by other applications because of its computational efficiency and robustness.
# TABLE OF CONTENTS

1 Introduction .........................................................................................7  
   1.1 Significance ..................................................................................7  
   1.2 Objectives and Methods ...............................................................8  

2 Background Studies .........................................................................10  
   2.1 Definitions ....................................................................................10  
   2.2 Motives for Mirroring .................................................................12  
   2.3 Classifications of Mirroring .........................................................13  
   2.4 Related Research on Mirroring ....................................................15  

3 System Overview ...............................................................................17  
   3.1 System Structure ..........................................................................17  
   3.2 Web Spiders ..................................................................................21  
   3.3 Mirror Hosts Detecting Processes ..............................................24  
   3.4 Database Connectivity ...............................................................25  
   3.5 JSWDK Server ..............................................................................25  
   3.6 Servlets .........................................................................................26  

4 A Mirror Detecting Algorithm .........................................................28  
   4.1 Candidate Pair Detection .............................................................28  
      4.1.1 Feature Generation ...............................................................29  
      4.1.2 Feature Matching .................................................................30  
   4.2 Host pair Classification ...............................................................32  
   4.3 Examples ......................................................................................35  

5 User Interface and Experimental Results .........................................43  
   5.1 User Interface ...............................................................................43  
   5.2 System API Documentation .......................................................46  
   5.3 URL Collecting .............................................................................48
5.4 Mirror Host Detecting ........................................................................51
  5.4.1 A Level-2 Mirror Hosts Testing ..............................................51
  5.4.2 A Level-4 Mirror Hosts Testing ..............................................57

6 Conclusions and Future Work .........................................................62
  6.1 Conclusions ..............................................................................62
  6.2 Future Work .............................................................................62

7 References ......................................................................................64
List of Figures

Figure 1  System flow chart…………………………………………………………19
Figure 2  System structure…………………………………………………………20
Figure 3  Web spider control flow chart………………………………………….23
Figure 4  The index web page of this project………………………………………44
Figure 5  Interface of the spider………………………………………………….44
Figure 6  Interface of the candidate mirror generation……………………………45
Figure 7  Interface of the classifier………………………………………………45
Figure 8  System API documentation……………………………………………47
Figure 9  The completeness of URL collecting of the spider……………………48
Figure 10 Collected URLs display…………………………………………………50
Figure 11 Similarity scores with different sift values……………………………54
Figure 12 Two Auburn University root pages at different time …………………56
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1</td>
<td>Similarity levels</td>
<td>15</td>
</tr>
<tr>
<td>Table 2</td>
<td>Classification of host pairs</td>
<td>35</td>
</tr>
<tr>
<td>Table 3</td>
<td>URL distribution in test 1</td>
<td>52</td>
</tr>
<tr>
<td>Table 4</td>
<td>Features generated in test 1</td>
<td>53</td>
</tr>
<tr>
<td>Table 5</td>
<td>Paths randomly selected in test 1</td>
<td>55</td>
</tr>
<tr>
<td>Table 6</td>
<td>Mirror sites of chip directory</td>
<td>58</td>
</tr>
<tr>
<td>Table 7</td>
<td>Chipdir URLs</td>
<td>59</td>
</tr>
<tr>
<td>Table 8</td>
<td>Features generated in test 2</td>
<td>60</td>
</tr>
<tr>
<td>Table 9</td>
<td>Paths randomly selected in test 2</td>
<td>61</td>
</tr>
</tbody>
</table>
1 Introduction

This chapter introduces the mirroring phenomenon on WWW and describes the significance of mirroring detection. The objectives of this project and the methods used are also described.

1.1 Significance

Web site mirroring has been introduced for reducing the access delay and providing load balancing in network servers. As the Internet is growing in a tremendous speed, the number of duplicated documents skyrocketed. Two previous studies, one from Digital Inc. [1], based on 30 million pages collected by the AltaVista search engine during 1996, and one from Stanford University [8], based on 26 million pages collected by the the Google search engine [10] during 1997, point out the large amount of page duplication on the Web. The Digital study reported 30% duplication, and the Stanford study about 36% duplication. Both studies identified mirroring, that is, the systematic replication of content over a pair of hosts, as the principal cause of duplication. In another word, web search engines are collecting about one third information they already know and will probably return a lot of links to users that contain very similar content.

The explosive growth of the World Wide Web (WWW) has created both challenges and opportunities for the information retrieval discipline. Searching for information on the WWW differs from information retrieval in classical collections in many respects. Two of the most obvious are the very large size of the Web, estimated at 275 million pages as of March 1998, and growing at the rate of 20 million pages a month [6], and the prevalence of systematic content duplication. The fraction of the total WWW
collection consisting of duplicates and near duplicates has been estimated at 30 to 45% [1, 8]. The principal reason for duplication on the Web is the systematic replication of content across distinct hosts, a phenomenon known as mirroring. It is estimated that at least 10% of the hosts on the WWW are mirrored [2, 4]. Duplication has both positive and negative aspects. On one hand the redundancy makes retrieval easier: if a search engine has missed one copy, maybe it has the other; or if one page has become unavailable, maybe a replica can be retrieved. On the other hand, from the point of view of search engines, storing duplicate content is a waste of resources and from the user's point of view, getting duplicate answers in response to a query is a nuisance.

Reliable algorithms for finding mirrors can improve web based information access in several ways: Although search engines tend to filter out exact duplicates and even some near duplicates, considerable duplicate content stored on mirrored hosts escapes detection. This wastes index space and annoys users. Mirror detection can alleviate this problem. Transient web server failures frustrate Web users. If a mirroring relationship is known for the failed server, such broken links can be made invisible by fetching the document from a mirrored host. Web browsers and web proxies cache documents to improve the speed at which information is accessed. A document in the cache can potentially be used to satisfy a request for the same path on a mirrored host as well.

1.2 Objectives and Methods

In this project, an algorithm for finding mirrored hosts on the Web is discussed and evaluated. This algorithm is a top-down algorithm, that is, it is based on page
attributes such as URL, IP address, and connectivity, and not on the page content. Essentially the replicated structure of mirrors is used to identify the mirrored hosts. The objective of this project is to present and implement an algorithm for detecting mirroring on the WWW in an information retrieval framework. This is based on the previous work done by Krishna Bharat and Andrei Broder at Compaq Research Center [4].

In this project, only incomplete sets of URLs of WWW hosts need to be collected. Features will be derived from the sets of URLs and the structural similarity can be calculated by comparing these sets of features. Once two hosts are identified as structural similar, a validation and classification step is taken. Source documents are retrieved from both hosts against the same URLs and the resemblance between these document pairs are compared to get a set of resemblance values. Based on a classification criterion that will be introduced later, the level of the mirroring extent of the two hosts is determined.

The organization of this paper is as following: Chapter 2 gives background studies of mirror hosts detecting. Chapter 3 describes the mirror hosts detecting system built in this project. The mirror-detecting algorithm is elaborated in Chapter 4. Chapter 5 shows the user interface and experimental results. Finally, we conclude this paper in Chapter 6.
2 Background Studies

In this chapter, we will present a brief review of background studies. First we present the related definitions. Then the motives for mirroring are listed. The classification method is illustrated after that. At last, related research on mirroring is introduced.

2.1 Definitions

Let’s start with some definitions. Each document on the WWW has a unique name called the Universal Resource Locator (URL). The URL consists of three disjoint parts, namely the access method, a hostname, and a path. For example in the URL
http://www.auburn.edu/~lianghu/ the string http defines the access method,
http://www.auburn.edu/ is the hostname, and ~lianghu is the path. The hostname identifies a host also known as a web server on which the document is stored. Sometimes it may also contain a port number. A host stores a collection of documents which all share the same hostname. Its path identifies each document on a host. The hostname can be translated by a name server into an Internet Protocol (IP) address represented as four octets (bytes). This relation is many to many, that is, several hosts with different hostnames can share the same IP address (virtual hosting) or a host may have a set of associated IP addresses, in which case the name server returns an arbitrary member of the set.

Two hosts are mirrors if they contain the same set of documents identified by the same paths. However, at one extreme one can say that two sites are mirrored if their content is byte-wise identical. In practice this definition is too restrictive: even on
successive accesses to the same URL the fetched content may differ slightly because of
dynamic components, timestamps, transaction-ids, etc. At the other extreme, one can say
that two sites are mirrors if enough pages on one are very similar to pages on the other.
This definition does not address the issue of structure. There may exist many different
websites extracting information from the same resource. But they are so different that I
can never say they are mirrored.

Hence, two hosts are mirrors if:

• A high percentage of paths (that is, the portions of the URL after the
  hostname) are valid on both web sites, and

• These common paths link to documents that have similar content.

Highly similar is a subjective measure. I made this notion precise by adopting the
resemblance distance described in [3] that experimentally captures well the informal
notion of roughly the same. The technique efficiently computes the syntactic resemblance
between two documents as a fractional score between 0 and 1. The higher the score is, the
greater the resemblance. Any distance measure that computes a similar resemblance score
can be substituted. One thing needs to be pointed out is that under this definition, hosts
replicate contents with different path names will not be considered as mirrors.

The technique we devised to detect mirrored hosts from large data sets depends
mostly on the syntactic analysis of URL strings and requires fetching and content
analysis only for a small number of pages. We used probabilistic tests for establishing the
degree of mirroring. This makes our technique computationally very efficient. We are
able to detect both partial and total mirroring, and handle cases where the content is not
exactly identical. Furthermore, this strategy does not assume that the initial set of URLs
from each host is comprehensive. Hence, the technique has practical uses beyond this study, and can be applied to other settings. For example, from the point of view of web crawlers and caching proxies, detecting mirrors can be valuable to avoid redundant fetching and knowledge of the existence of mirrors can be used to compensate for broken links.

2.2 Motives for Mirroring

There are many incentives to replicate data on the web:

- **Load Balancing**: Replication decreases server load.

- **High Availability**: The Protein Data Bank example presented in Section 4.2 is an instance of geographical mirroring for high availability.

- **Multi-lingual Replication**: The same data is made available in many languages. For example, several Canadian sites have mirrors that only differ in the language used, French or English.

- **Franchises/Local Versions**: This happens when content is licensed to other parties. For example, http://quicken.excite.com and http://quicken.com have the same content, except for some site-specific branding.

- **Database Sharing**: Two independent sites may share the same database or file system leading to mirroring, unintentionally.

- **Virtual Hosting**: These represent services that use the same IP address (and hence the same server) but implement two different web sites based on the host name in some cases paths on one are valid on the other and yield identical pages. This need not always be so. In other cases the two virtual
hosts share the same IP address but paths on one may not be valid on the other, and common paths lead to dissimilar content.

- **Maintaining Pseudo Identities:** Often, the incentive for such replication is to spam search engines with seemingly different web sites that in fact have the same content, hoping that one of them gets listed at the top of the ranking order.

### 2.3 Classification of Mirroring

There exist various degrees of mirroring. To classify them, it is helpful to distinguish between structure and content.

- Structure is defined by the set of valid paths relative to the host under consideration. If two hosts have exactly the same set of paths, they are structurally identical.

- Content is defined by the non-formatted text of each web page stored on the host. Two pages are content-identical if they are byte-wise equal.

However during mirroring, pages often change at the byte level (e.g., by the addition of blank lines, by HTML reformatting, etc) without any change of content. Hence two pages are *content equivalent* if they have identical content after such normalizations. If pages change in content (e.g., due to a banner ad or other forms of dynamic content) but remain highly similar at the syntactic level, I call them *highly similar*. High similarity may be defined based on the edit distance or some other suitable measure. This paper uses the similarity measurement technique [1], with a suitable threshold. Finally, if pages change
substantially at the syntactic level but are semantically similar (e.g., translated content), I call them related. This leads to the following classification of mirroring levels:

Level 1: Structural and content identity.

Every page on host A with relative path P, (i.e., a URL of the form http://A/P) is represented by a byte-wise identical page on host B, at location http://B/P, and vice versa.

Level 2: Structural identity. Content equivalence

Every page on host A with relative path P is represented by an equivalent content page on host B, at location http://B/P, and vice versa.

Level 3: Structural identity. Content similarity.

Every page on host A with relative path P is represented by a highly similar page on host B, at location http://B/P, and vice versa.

Level 4: Partial structural match. Content similarity.

Some pages on host A with relative path P, are represented by a page on host B, at location http://B/P, and vice versa, and these pairs of pages are highly similar

Level 5: Structural identity. Related content.

Every page on host A with relative path P is represented by a page on host B, at location http://B/P, and vice versa. The pages are pair-wise related (e.g., every page is a translation of its counterpart) but in general are not syntactically similar.

Mismatch -- None of the above.

This induces the partial order shown below. Level 4 and Level 5 are not comparable.
2.4 Current Research on Mirroring

Several researches on mirroring have been conducted before. Bharat and Broder compared several algorithms for identifying mirrored hosts on the World Wide Web [5]. The algorithms operate on the basis of URL strings and linkage data: the type of information easily available from web proxies and crawlers. They evaluated 4 classes of top-down algorithms for detecting mirrored host pairs (that is, algorithms that are based on page attributes such as URL, IP address, and connectivity, and not on the page content) on a collection of 140 million URLs (on 230,000 hosts) and their associated connectivity information. There is an alternative bottom-up approach presented by Cho, Shivakumar, and Hector Garcia-Molina in [3] whereby in the first stage of the algorithm
copies of a given pages are clustered together, and then these clusters are grown until they represent an entire site.

Beyond finding mirror sites, some researchers tried to make best performance out of mirror sites. One promising approach is accessing mirror sites in parallel. Andreas found that dynamic parallel access offers transmission rates at least as high as the fastest host [9]. Dinda concluded that clients wishing to achieve near-optimal downloading performance might only need to consider a small number of mirror hosts rather than all mirrors [7].

In summary, the amount of duplicate web pages because of mirroring is not negligible and could cause trouble for web search engines and web caches. The motives of mirroring include load balancing, high availability, multilingual replication, franchises/local version, database sharing, virtual hosting, and maintaining identity. According to the similarity of structure and contents, mirroring of web sites is classified into 5 different categories. Finding mirror sites and then making the best of using these mirror sites are an interesting research area.
3 System Overview

A prototype of mirror site detecting system is designed and developed. It is composed of a web spider, a mirror-detecting kernel, a relational database, and web interface. It utilizes the technologies of JavaServer Web Development Kit (JSWDK) and Servlets. The system can be accessed on-line at

http://lab40.eng.auburn.edu:8080/msproject/.

3.1 System Structure

The system flow chart is shown in Figure 1. The spider receives a seed URL and starts traversing the web to collect URLs. The candidate mirror generation process generates features from these URLs, predicts the possible mirror sites by comparing the features and saves the mirror sites. The task of validation and classification process is to find out if the candidate mirror sites are valid. If they are, the levels of their similarity are calculated and saved for future use. These processes are Java processes running on the server side.

Structurally, the system could break into 3 tiers: client browsers, middle tier including servlets and server side programs, and backend database (Figure 2). The client browsers send HTTP requests through GET or POST method to the server. When the requests arrive at the server, the appropriate servlets are called. The doGet and doPost methods in servlets will extract information from the request object. The information will be used to call other server side programs or access database via JDBC. After the results are generated, the servlet will compose a new web page and send it back to the client browser.
To users, all the above processes are transparent. To use the system, users simply send command from client side web interfaces to the server. The corresponding results will be generated by JSWDK server and sent back to client side. Each part of this system will be explained in detail in the following sections.
Figure 1: System flow chart
Figure 2: System structure
3.2 Web Spiders

Web spiders, also known as web crawlers, robots, or wanderers, are software programs that automatically traverse the web. Search engines use spiders to find what’s on the web; then construct an index of the pages that were found. In this project, collecting URLs from the web is the first step. Thus, a web spider was created to fulfill this task.

The spider starts by parsing a specified web page, root page, noting any hypertext links on that page that point to other web pages. It then parses those pages for new links, and so on, recursively. Spider software doesn’t actually move around to different computers on the Internet, as virus or intelligent agents do. A spider resides on a single computer. It simply sends HTTP requests for documents to other machines on the Internet, just as a web browser does when users click on links. All the spider really does is to automate the process of traversing links.

Due to the time limit and hardware constraint, a thorough traversal of the WWW is extremely difficult, if not impossible. So, the spider of this project performs a breadth first search. The intention is once a web host is reached, we want to collect as many as possible URLs from that host that is concerned. Considering the tree like structure of the web, a queue was utilized to accomplish breadth first search. The FIFO feature of queue will ensure that all the links inside a specific document are collected before the spider goes out. All the URL links are put into the queue, waiting for being parsed. To prevent infinite loop, a separate data structure, a set, is used to save all the visited URLs. A URL can only be put into the queue when it is not visited before. A URL link could point to any kind of file. In this project, only “text/html” file type is interested.
As shown in Figure 3, a thread of program keeps reading URLs from the queue and retrieves document to which the URL points. Then all the URLs in the document will be put into the queue for later processing. The thread is killed until the queue is empty or the search limit is reached. Multiple threads of spider can run simultaneously to increase the speed of collecting.

This spider is able to handle parsing frameset pages. It also respects the robots exclusion standard, meaning that it avoids sites where spiders are not welcome. Any site can exclude web spiders from all or part of its file system by putting certain statements in a file called robots.txt. Before retrieving a document, the spider will look at the robots file at first. The spider is implemented using JSDK1.2.2 on Solaris 5.6 platform.
Figure 3: Web spider control flow chart

1. Insert seed URL into a queue.
2. While the queue is not empty, get a URL from the queue.
3. Is robot welcome?
   - N
   - Y
     - Retrieve the source file.
4. Is it FrameSet?
   - Y
      - Use “src” as link indicator.
   - N
5. While not end of file, get next links.
6. Is it text/html?
   - Y
     - Has not visited?
       - Y
         - Put into the queue and db
       - N
   - N

Figure 3: Web spider control flow chart
3.3 Mirror Hosts Detecting Processes

The mirror detecting processes are Java programs residing on the JSWDK server. They inherit from HttpServlet class and override the doGet and/or doPost method(s), so that they are able to receive requests from and send responses to client browsers. Given a set of URLs collected by the web spider, they can detect pairs of mirrored host. The URL set needs not to be comprehensive. The mirror detecting processes are responsible of returning reasonable results based on a probabilistic theory. There are 3 Java programs to accomplish the job:

- CandidateGenerator
- ShingleTest
- Classifier

The CandidateGenerator generates candidate mirrored host pairs from the URL set by syntactic analysis. What it does is to cut the URL strings into single words and use the word bi-grams to predict the similarity of site structure. Word bi-grams are 2 consequent words in a string. More same word bi-grams in common means greater possibility of being mirrored host. The Classifier will take the candidate mirrored host pairs to test if they are real mirrored hosts and classify the extent they are mirroring. To do this, it needs to randomly select paths to fetch documents from both hosts and compare the shingles derived from the documents. In Chapter 4, classification criterion will be described. The ShingleTest provides utility functions of making shingles out of documents.

The communication between CandidateGenerator and Classifier is through a temporary database table. The candidate host pairs are saved into this table at the end of CandidateGenerator process and read out from it by Classifier process. Database access
overhead is an issue here. There exists a tradeoff between task decomposition and inter-process communication, which is beyond the concern of this project. The algorithm behind the scene is very complicated. The algorithm and implementation will be addressed in detail in Chapter 4.

3.4 Database Connectivity

Oracle 8i is the backend DBMS of this project. A relational database is created to store intermediate and final results. The URL table is used to save URLs collected by the spider. The mirror table is used to save mirror sites and their classifications.

JDBC 2.0 was employed by the system to communicate with the database. JDBC technology is an API that accesses virtually any tabular data source from the Java programming language. It provides cross-DBMS connectivity to a wide range of SQL databases. In simplest terms, a JDBC driver makes it possible to do three things:

- Establishes a connection with a database
- Sends queries and update statements to the database
- Processes the results

3.5 JSWDK Server

To bring this project online, a web server that supports servlets is used. JSWDK was chosen because it is free, easy to install and easy to use. The JavaServer™ Web Development Kit (JSWDK) is the reference implementation for JSP and the Java™ Servlet API. This release of JSWDK supports JSP 1.0 and the Servlet API 2.1.
The JSWDK contains a simple servlet engine for developing and testing servlets, the `javax.servlet` and `javax.servlet.jsp` package sources (which are the JSP and Servlet APIs), and API documentation. The JSWDK also contains a simple HTTP web server and a JSP-enabled engine. After downloading JSWDK package from `http://java.sun.com/products/jsp/archive.html`, it was extracted and installed to an engineering user space. The detailed installation and configuration procedures are available on `http://lab40.eng.auburn.edu:8080/`. The server is a small java program. To start it, just use command `startserver` in the shell under the directory where the JSWDK root resides.

### 3.6 Servlets

Servlets are the Java platform technology of choice for extending and enhancing web servers. Servlets provide a component-based and platform-independent method for building web-based applications, without the performance limitations of CGI programs. And unlike proprietary server extension mechanisms (such as the Netscape Server API or Apache modules), servlets are server- and platform-independent. This leaves us free to select a “best of breed” strategy for our servers, platforms, and tools.

Servlets have the accesses to the entire family of Java APIs, including the JDBC API to access databases. Servlets can also access a library of HTTP-specific calls and receive all the benefits of the mature Java language, including portability, performance, reusability, and crash protection. Servlets are a popular choice for building interactive web applications.
Servlets make use of the Java standard extension classes in the packages
`javax.servlet` (the basic Servlet framework) and `javax.servlet.http` (extensions of the
Servlet framework for Servlets that answer HTTP requests). In this project all servlets
extend HttpServlet because HTTP request and response will be manipulated. All servlets
must have `doGet` or `doPost` methods. They are responsible for handling the HTTP GET
or POST requests. These two methods take two objects as arguments: `HttpServletRequest`
and `HttpServletResponse`. The `HttpServletRequest` contains information from HTML
forms and the `HttpServletResponse` send the results back to the client browser.

In this project, the Spider, the CandidateGenerator, and the Classifier are
themselves servlets. They take parameters from user interface and execute, then compose
a new result page and display it to the user. To accomplish their job, servlets may call
other server side programs. For example, the SpiderServlet needs to call SpiderBean to
traverse the WWW. SpiderServlet only tells the users what’s going on and shows the
results. The advantage of doing this kind of OO design is to make the development and
maintenance easy and simple.
4 A Mirror Detecting Algorithm

The algorithm for detecting mirrored host pairs consists of two stages:

**Stage I: Candidate Pair Generation.** Considers the URLs available from each of the hosts in the database. The output from this stage is a list of host pairs that are potential mirrors, ordered by likelihood.

**Stage II: Host Pair Classification.** Processes the list of candidate host pairs from Stage I, and tests in each case if the hosts are indeed mirrors and estimates the extent of their duplication.

The tests are probabilistic in nature. That is, it is conceivable that it may fail to recognize a pair of mirrored hosts in Stage I, or it may make mistakes in measuring the degree of overlap between them in Stage II.

4.1 Candidate Pair Generation

This stage computes a set of features for each host, and searches for pairs of hosts that share many features in common. Using full paths as features is not practical, since we assume that our information with respect to any given host is incomplete (i.e., not all paths are available). Instead we use fragments of the path and the hostname as features, because URL paths usually represent the directory structure within which the web pages are located. Consequently, we can expect prefixes representing top-level directories to be shared by many URLs. Pairs of consecutive directories' names along the path (called word bi-grams) are used as features.
Several ways are taken to reduce the number of features. One way to do this is to ignore hosts that contribute a small number of URLs to the collection. Such sites are either too small in size to have much mirrored content, or poorly sampled, in which case the samples in the collection may not be very representative. The threshold value of URL number is arbitrary. A further data reduction is obtained as follows: for every site we first sort the list of URLs, and then only consider those paths whose strings upon hashing yielded a value which is 0 mod m. The aim of this step is to increase the correlation between the selected paths. If a path is selected on one host, it increases the likelihood that the same path is selected on its mirror.

4.1.1 Feature Generation

After preparation, each URL string is converted to a list of terms by:

- Converting to lowercase. Paths may or may not be case sensitive, but hostnames are not.

- Treating sequences of non-alphabetical characters as word-breaks. This gives a list of words. For example, www1.edu and www2.edu will yield the same list: (www,edu).

The word bi-grams are generated by treating every contiguous pair of terms as features. For example, the hostname http://www.auburn.edu/ generates

(www,auburn) and (auburn,edu),

Depth information is also recorded with path fragmentations. This gives positional word bigrams. For example:

/csse/people/grad/user/index.html
gives

\[(csse, people, 0)(peole, grad, 1)\]

\[(grad, user, 2)(user, index, 3)(index, html, 4)\]

as features. Position is useful because this algorithm tries to find mirror sites that share the same path structure. Finding mirror sites that both mirror content and rename paths is a harder problem, which will not be addressed in this project.

### 4.1.2 Feature Matching

The tuple \(<feature, host>\) stores each feature and its associated host. At the end of the run, the tuples are sorted by the first element of the tuple, namely the feature. For any given feature, this causes all hosts that contain the feature to be listed contiguously. This allows the list to be processed in a single pass in order to compute the similarity between pairs of hosts that share common features.

In our feature-matching scheme, features are weighted in inverse proportion to the number of hosts they occur in. This corresponds to the usual IDF (inverse document frequency) weighting used in information retrieval. IDF is a measure of how often a particular term appears across all of the documents in a collection. It is usually defined as

\[\log\text{(collection size/number of documents containing the term)}.\]

So common words will have low IDF and words unique to a document will have high IDF. Features that occur in many hosts are considered too common to be significant, and correspondingly their IDF weight is low. As an optimization, all features that occur within more than 20 hosts are ignored. For the rest, for every feature \(f\), we consider each pair of hosts that share it and increment the Similarity\_Score(Host\(_1\), Host\(_2\)) by the weight of the feature, \(FW(f)\).
FW(f) is defined as:

\[ FW(f) = \frac{S(f)}{N(f)} , \]

where \( S(f) \) is a measure of the significance of \( f \) independent of its distribution, and \( N(f) \) is the number of hosts in which \( f \) occurs. \( S(f) \) is set to 4 for host features and to 1 for path features, since a host feature match offers evidence that the two sites are part of the same organization (given that I consider only rare features).

The Normalized Similarity Score of each pair (Host\(_1\), Host\(_2\)) is computed thus:

\[
\text{Similarity\_Score(Host}_1, \text{Host}_2) = \text{Normalized\_Score(Host}_1, \text{Host}_2) = \frac{1 + 0.1( \log(\text{N}_1) + \log(\text{N}_2) )}{\text{Log}(\text{N}_1) + \log(\text{N}_2)}
\]

\( \text{N}_1 \) and \( \text{N}_2 \) represent the number of URLs in the input from Host\(_1\) and Host\(_2\) respectively, and are assumed to be proportional to their sizes. The denominator helps normalize the score by host size, to compensate for the fact that large hosts will have more feature matches than small hosts. These parameters appear to perform satisfactorily but may afford some tuning.

At the end of this run, the normalized similarity score of every pair of hosts that share a significant feature will be computed. A list of <score,host-pair> tuples is saved. The remaining host pairs are sorted in the descending order of normalized similarity score. A list of prospective mirroring hosts, in the decreasing order of probability is obtained as the result of stage I.
4.2 Host Pair Classification

In stage II, the list of sorted host pairs from the previous stage is processed, and each host pair is classified into one of many categories based on the classification of mirroring hosts discussed in previous section.

To classify a host pair we estimate the fraction of the paths from one host that are valid on the other host, and the extent to which the pages referenced by the common paths are similar. Thus, there are two steps: (1) selecting paths to test, and (2) checking for validity of paths and computing similarity between pages corresponding to the valid paths.

The first step is done only once per host. Since a given host may occur in many host pairs, the same paths are reused. For each pair of host pages, the two root pages (the null path) are always considered. Also, in the experiment, 9 additional paths are selected at random per host from the paths chosen in Stage I, giving a total to 10 path samples per host.

In the second step the selected paths are used to fetch documents from both hosts, namely the source host where the path is known to be valid, and the target host, where the validity of the path is to be tested. The GET could fail at the source or at the target. If fetching both pages is successful, a further check is performed to see if the contents are the same. This can be done at various levels of tolerance to change. In the strictest instance, this can be done by fingerprinting both documents (i.e., computing a checksum, such that with high probability any two distinct documents will have different checksums). However, in practice content is often non fingerprint-identical. This can
happen even on successive queries to the same server, because of variable server-side includes or dynamically generated web pages. With mirrored content there is an even greater likelihood of a discrepancy, due to version inconsistencies and local server-side includes. To compensate for this I de-tag the content, ignore white space and check for syntactic similarity, that is, closeness of textual content, instead of fingerprint equality. Following [3] I use the mathematical concept of "resemblance" to capture the informal notion of syntactic similarity.

We view each document as a sequence of words, and start by lexically analyzing it into a canonical sequence of tokens. This canonical form ignores minor details such as formatting, html commands, and capitalization. We then associate with every document \( D \) a set of subsequences of tokens \( S(D, w) \).

A contiguous subsequence contained in \( D \) is called a shingle. Given a document \( D \) we define its \( w \)-shingling \( S(D, w) \) as the set of all unique shingles of size \( w \) contained in \( D \). So for instance the 4-shingling of \( \langle \text{a, rose, is, a, rose, is, a, rose} \rangle \) is the set

\[
\{ \langle \text{a, rose, is, a} \rangle, \langle \text{rose, is, a, rose} \rangle, \langle \text{is, a, rose, is} \rangle \}
\]

The larger the shingle size is, the stricter the matching rule. In this project, users are allowed to input different shingle size.

The resemblance, \( r(A, B) \), of two documents \( A \) and \( B \) is defined as follows: first each document is transformed into a set of word k-grams, \( S(A) \) and \( S(B) \) also called "shingles". Then I compute

\[
r(A, B) = \frac{|S(A) \cap S(B)|}{|S(A) \cup S(B)|}.
\]

where \(|S|\) is the size of the set \( S \).
The resemblance can be estimated using a fixed size “sketch” for each document. To reduce CPU time, we don’t have to consider all the shingles of a document. Use modulus method to select only those shingles whose hash values are 0 mod the sketch value. The sketch value is also an arbitrary value that can be determined by users. Clearly, the resemblance value, \( r(A,B) \) is a number between 0 and 1, which can be used to express the similarity between A and B as a percentage.

By considering structural resemblance and/or content resemblance (with different similarity thresholds), the host pairs were classified into one of many levels. For our experiment I defined the threshold for high similarity as 50\%. Additionally, I used a 100\% threshold to check for full similarity and a 0\% threshold for trace similarity. When all similarity checks fail, the pair is classified as a Mismatch.

Between any pair of hosts there are 19 web page comparisons in all: a pair of root pages + 9 source pages sampled from each host. In each case one of the following outcomes is possible

- **SF**: Source Failure. GET failed at source host.
- **TF**: Target Failure. GET failed at target host.
- **FM**: Fingerprint Match. Content is byte-wise identical.
- **FS**: Full similarity. The documents are 100\% equivalent after de-tagging.
- **HS**: High similarity. Common content is above the threshold for high similarity.
- **TS**: Trace Similarity. A small (non-zero) portion of the document is common.
- **NS**: Path is valid, but no syntactic similarity.

Based on the results of the 19 tests, each host pair is assigned a classification level. The assignment criteria for various classification levels is summarized below:
<table>
<thead>
<tr>
<th>Classification</th>
<th>Criterion</th>
<th>Implication</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 1</td>
<td>All tests show SF or FM and not all are SF.</td>
<td>Same Structure. Identical Content.</td>
</tr>
<tr>
<td>Level 2</td>
<td>All tests show SF or FM or FS, and not all are SF</td>
<td>Same Structure. Equivalent Content.</td>
</tr>
<tr>
<td>Level 3</td>
<td>All tests show SF or FM or FS or HS, and not all are SF.</td>
<td>Same Structure. Similar Content.</td>
</tr>
<tr>
<td>Level 4</td>
<td>Some tests show HS or FM.</td>
<td>Structure is partially replicated. For replicate paths, content is similar.</td>
</tr>
<tr>
<td>Level 5</td>
<td>No test yields TF and at least one with TS.</td>
<td>Same structure (since all paths are valid). Some of the content appears related.</td>
</tr>
<tr>
<td>Mismatch</td>
<td>All tests result in TF or NS.</td>
<td>Not similar</td>
</tr>
</tbody>
</table>

Table 2: Classification of host pairs

### 4.3 Examples

The following example illustrates the algorithm. The web site of Oracle company has two domain names: http://www.oracle.com/ and http://bigip-www.us.oracle.com/. These two DSN names are aliases to each other, i.e., they are fully mirroring. We will show step by step how to find out that they are fully mirroring to each other by using our algorithm.

**Stage I: Candidate Pair Generation**

32 URLs are collected from http://www.oracle.com/ and 31 URLs are collected from http://bigip-www.us.oracle.com/. In preparation, we use 7 as the sift value to reduce
the number of features. Only the paths whose strings upon hashing yield a value that is 0 mod 7 are considered. In the collected URLs, only three paths have hash value that is 0 mod 7.

<table>
<thead>
<tr>
<th>Path</th>
<th>Hash Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>/corporate/contact/index.html?content.html</td>
<td>-1156892688</td>
</tr>
<tr>
<td>/corporate/news/index.html?content.html</td>
<td>-1866281739</td>
</tr>
<tr>
<td>/ip/</td>
<td>1504601</td>
</tr>
</tbody>
</table>

Therefore, the following URLs are chosen to perform the test.

http://bigip-www.us.oracle.com/ip/
http://www.oracle.com/ip/

These URLs are used to generate features. Each URL is decomposed into words. All the non-alphabetic characters will be deemed as delimiters. For example,


will be decomposed into

(bigip, www, us, oracle, com, corporate, contract, index, html, content, html).

Every two consequent words and their position in the URL will be composed as a feature.
Host features always have position –1. Path features’ positions are determined by the first word’s position in the path. For example, \((\text{bigip, www, -1})\) is a host feature, while \((\text{corporate, contact, 0})\) and \((\text{contact, index, 1})\) are path features. As a result, a set of feature-host tuples is produced as following:

1. \(\langle \text{(bigip,www,-1)}, \text{bigip-www.us.oracle.com} \rangle\)
2. \(\langle \text{(contact,index,1)}, \text{bigip-www.us.oracle.com} \rangle\)
3. \(\langle \text{(contact,index,1)}, \text{www.oracle.com} \rangle\)
4. \(\langle \text{(corporate,contact,0)}, \text{bigip-www.us.oracle.com} \rangle\)
5. \(\langle \text{(corporate,contact,0)}, \text{www.oracle.com} \rangle\)
6. \(\langle \text{(corporate,news,0)}, \text{bigip-www.us.oracle.com} \rangle\)
7. \(\langle \text{(corporate,news,0)}, \text{www.oracle.com} \rangle\)
8. \(\langle \text{(html,content,3)}, \text{bigip-www.us.oracle.com} \rangle\)
9. \(\langle \text{(html,content,3)}, \text{www.oracle.com} \rangle\)
10. \(\langle \text{(index,html,2), bigip-www.us.oracle.com} \rangle\)
11. \(\langle \text{(index,html,2)}, \text{www.oracle.com} \rangle\)
12. \(\langle \text{(news,index,1), bigip-www.us.oracle.com} \rangle\)
13. \(\langle \text{(news,index,1)}, \text{www.oracle.com} \rangle\)
14. \(\langle \text{(oracle,com,-1)}, \text{bigip-www.us.oracle.com} \rangle\)
15. \(\langle \text{(oracle,com,-1)}, \text{www.oracle.com} \rangle\)
16. \(\langle \text{(us,oracle,-1), bigip-www.us.oracle.com} \rangle\)
17. \(\langle \text{(www,oracle,-1)}, \text{www.oracle.com} \rangle\)
18. \(\langle \text{(www,us,-1)}, \text{bigip-www.us.oracle.com} \rangle\)
In this set of features, http://www.oracle.com/ and http://bigip-www.us.oracle.com/ share 1 host feature and 6 path features. The similarity score of this host pair is the sum of INF weight of each feature. In this case, the path feature has a weight of $1/2 = 0.5$, and the host feature has a weight of $4/2 = 2$. Therefore, the similarity score is $(2 + 0.5 \times 6) = 5$.

By applying the normalization function mentioned before, the normalized similarity score is

$$\frac{5}{1 + 0.1 \times (\log N1 + \log N2)} = \frac{5}{1 + 0.1 \times (\log 32 + \log 31)} = 3.9$$

This is the end of Stage I processing, which returns a normalized similarity score of the host pair, http://www.oracle.com/ and http://bigip-www.us.oracle.com/. This similarity score suggests that these two hosts are probably mirroring each other. In order to further validate this suggestion and classify the extent of the mirroring, Stage II processing is conducted.

**Stage II: Host Pair Classification**

There are two steps in Stage II processing: (1) selecting paths to test, and (2) checking for validity of paths and computing similarity between pages corresponding to the valid paths. In the first step, we randomly select 9 paths from each host. The 9 paths are used to fetch pages from the host itself and the other host.

For example, the path /corporate/news/ is used to fetch pages from both http://www.oracle.com/corporate/news/ and http://bigip-www.us.oracle.com/corporate/news. The two pages fetched from both hosts are compared to each other and a resemblance value is generated for the two documents. In this particular case, all the documents retrieved from both hosts are Fingerprint Match (FM). From Table 2, it implies that http://www.oracle.com/ and http://bigip-www.us.oracle.com/ have the same structure and identical content. They are level-1 mirroring hosts. This shows that our algorithm is able to detect the level-1 mirroring hosts.

**Document Resemblance**

In stage II, if two documents are not FM, we have to use shingles and sketches to compute their resemblance. In Chapter 5, we will give more examples on level-2 and level-4 mirroring. Here we only give an example of how to compute the document resemblance.
The following shows how to compare the resemblance of


file2.html. The source code of http://www.auburn.edu/~lianghu/file1.html is:

<html><body>
This is the first file that is used for the document resemblance test.
</body></html>

and the source code of http://www.auburn.edu/~lianghu/file2.html is:

<html><body>
This is the second file that is used for the document resemblance test.
</body></html>

All characters in the documents are transformed into lowercase ones and all the
HTML tags are removed. Then, each document is decomposed into a sequence of words.
A set of shingles is generated from each of the two sequences of words. For example, the
shingles (size = 5) of http://www.auburn.edu/~lianghu/file1.html, S(A), are

<table>
<thead>
<tr>
<th>Shingle</th>
<th>Hash Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. [this, is, the, first, file]</td>
<td>194081744</td>
</tr>
<tr>
<td>2. [is, the, first, file, that]</td>
<td>1517931795</td>
</tr>
<tr>
<td>3. [the, first, file, that, is]</td>
<td>1256383167</td>
</tr>
<tr>
<td>4. [first, file, that, is, used]</td>
<td>-1567285235</td>
</tr>
<tr>
<td>5. [file, that, is, used, for]</td>
<td>-1947881366</td>
</tr>
<tr>
<td>6. [that, is, used, for, the]</td>
<td>294036129</td>
</tr>
<tr>
<td>7. [is, used, for, the, document]</td>
<td>-338821825</td>
</tr>
</tbody>
</table>
8. [used, for, the, document, resemblance] \hspace{1cm} 1557940938
9. [for, the, document, resemblance, test.] \hspace{1cm} 455481165

and the shingles of \(http://www.auburn.edu/~lianghu/file1.html\), \(S(B)\), are

<table>
<thead>
<tr>
<th>Shingle</th>
<th>Hash Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. [this, is, the, second, file]</td>
<td>-856474996</td>
</tr>
<tr>
<td>2. [is, the, second, file, that]</td>
<td>-984556073</td>
</tr>
<tr>
<td>3. [the, second, file, that, is]</td>
<td>988670587</td>
</tr>
<tr>
<td>4. [second, file, that, is, used]</td>
<td>-1276440623</td>
</tr>
<tr>
<td>5. [file, that, is, used, for]</td>
<td>-1947881366</td>
</tr>
<tr>
<td>6. [that, is, used, for, the]</td>
<td>294036129</td>
</tr>
<tr>
<td>7. [is, used, for, the, document]</td>
<td>-338821825</td>
</tr>
<tr>
<td>8. [used, for, the, document, resemblance]</td>
<td>1557940938</td>
</tr>
<tr>
<td>9. [for, the, document, resemblance, test.]</td>
<td>455481165</td>
</tr>
</tbody>
</table>

The hash values are used to represent shingles. Usually a sketch of shingles will be
enough, which means only those shingles whose hash values are \(0 \mod \text{a sketch value}\)
will be considered. In this simple example, there are only 9 shingles. We’ll omit this step.

The shingles that are shared by both documents are the intersection of the two shingle
sets, \(S(A) \cap S(B)\). The intersection is

\((-1947881366, 294036129, -338821825, 1557940938, 455481165)\).
The unique shingles comprises the union of two shingle sets, \( S(A)US(B) \). The union is
\[
(-1947881366, 294036129, -338821825, 1557940938, 455481165, -1567285235, \\
1517931795, 194081744, 1256383167, -856474996, 988670587, -984556073, \\
-1276440623).
\]

The resemblance value of \( \text{http://www.auburn.edu/~lianghu/file1.html} \) and \( \text{http://www.auburn.edu/~lianghu/file2.html} \) is calculated as the size of the intersection divided by the size of the union.

\[
\frac{|S(A) \cap S(B)|}{|S(A)US(B)|} = \frac{5}{13} = 38.5\%
\]

As described in Section 4.2, resemblance below 50% belongs to TS (Trace Similarity). This implies that a small portion of \( \text{http://www.auburn.edu/~lianghu/file1.html} \) and \( \text{http://www.auburn.edu/~lianghu/file2.html} \) is common. In Stage II, once all the document pairs are compared and results are obtained, classification of the mirroring could be made by referencing Table 2.
5 Web Interface and Experimental Results

This chapter describes the web interface and API documentation for this project. Some experimental results of our mirror detection system are also presented.

5.1 Web Interface

To facilitate user control and information display, a user-friendly web interface is created by using HTML. Its URL is http://lab40.eng.auburn.edu:8080/msproject. From the HTML forms, servlets are called by setting attribute methods and actions. The most widely used methods to send HTTP requests are GET and POST. With a GET request the parameters are encoded in the URL, with a POST request they are transmitted in the request body. The difference makes them useful in different perspective. Since the parameters are included in the URL, the result page of GET request can be bookmarked and fetched again. On the other hand, the POST request is suitable for sending large file and confidential information. Either method works for this project. The action attribute informs the server which servlets should be called to handle this request. For example, to start a thread of web spider program, the following script is used:

```html
<form method = "get" action = "/servlet/mirror.SpiderServlet">
</form>
```

On the server side, all working servlets have to be placed in a folder named “servlet.”
There are three major interface that can be accessed via the main index page (Figure 4), and the API documentation links to a set of web pages that contain documentation generated by using a tool: javadoc. Interface 1 controls the web spider (Figure 5). Users can type in seed URL, set a search limit, select document type, start or stop the spider, list downloaded URLs, and delete the URLs from the database.

From interface 2, users can activate the server-side processes to generate all the candidate mirror host pairs (Figure 6). There are two parameters could be set. One is the
minimum number of URLs that a host should have in order to be considered as a potential mirror host. The other is the data reduction sift value. Only those paths whose strings upon hashing yielded a value which is $0 \mod \text{sift value}$ are used to generate features.

![Candidate Mirror Generation Interface](image)

Figure 6: Interface of the candidate mirror generation

Interface 3 fires the mirror validation and classification process (Figure 7). The sketch value is the number that will be used to reduce the number of shingles. Only shingles whose hash value is $0 \mod \text{sketch value}$ are used to calculate similarity. The shingle size is the number of words within a shingle. The larger it is, the more difficult it is to match two documents.

![Classifier Interface](image)

Figure 7: Interface of the classifier
5.2 System API Documentation

In order to make this mirror detecting system easy to understand and be available to other researchers who are interested in this topic, API documentation was created using javadoc and put online at: http://www.eng.auburn.edu/~lianghu/mirror/. Javadoc is the tool from Sun Microsystems for generating API documentation in HTML format from doc comments in source code. Doc comments are the special comments in the Java source code that are delimited by the /** … */ delimiters. These comments are processed by the javadoc tool to generate the API docs.

The generated HTML files are hierarchically structured. A style sheet file controls the look and feel of the doc web pages uniformly. They look exactly like the official Java API specification pages. With a plenty of hyperlinks, users can easily get class and method information in detail. The API documentation is a very important part of this project.
Figure 8: System API documentation
5.3 URL Collecting

To roughly test the completeness of URL collecting of our spider, web sites with different sizes were traversed. The result is shown in Figure 9. The number of URLs represents the size of a web site. The completeness is calculated as the number of collected URLs over the actual number of URLs. The traversal result indicates that as the sizes of web sites increase, the completeness of URL collecting decreases. URLs in small web sites with less than 20 can usually be completely collected (100%). 91% URLs can be collected if the site size is 20 to 50, and 88% for web sites with 50 to 100 URLs.

For large web sites with more than 100 URLs, the completeness drops to 74%.
The spider carries out a breadth first search. It collects as many URLs as possible within a web page before exiting. The completeness of collecting URLs from a host is not guaranteed. Some pages may be protected against robots, or need username and password to view. These URLs are beyond the reach of our spider. Fortunately, the mirror-detecting algorithm itself is probabilistic and is able to handle an incomplete set of URLs.

To check the searching results of the spider, the collected URLs need to be viewed. Click “List All” or “List a Site” button on the interface of spider (Figure 5), a web page will be generated and returned. The URL set might be too large to be shown in one page. This problem is solved by displaying a certain number of URLs on a single page. A page number index is provided at the bottom of each page to let users navigate through all the result pages (Figure 10). To achieve this, I used rownum, one of the pseudo columns in Oracle system. For example, to select the first 10 records from urls table, the SQL is like this:

\[
\text{select url from urls where rownum} \leq 10;
\]

However, to select the records from row 11 to row 20 will not work if you simply write:

\[
\text{select url from urls where rownum between 11 and 20;}
\]

This is because rownum must begin with 1. Therefore, we have to use a sub query in this case.

\[
\text{select url from urls where rownum} \leq 10 \text{ and url not in (}
\text{select url from urls where rownum} \leq 10);
\]
Figure 10: Collected URLs display
5.4 Mirror Hosts Detecting

Two tests were conducted to show our system’s capability of detecting mirror hosts. In test 1, we try to show that http://131.204.2.37/ and http://www.auburn.edu/ are mirror hosts with knowing they are actually the same host. In test 2, we will see how the system detects partial mirror hosts.

5.4.1 A Level-2 Mirror Hosts Testing

The purpose of this project is to detect mirror hosts. Taking a set of URLs collected from two hosts that are mirror to each other, this system should be able to detect the existence of the mirroring and classify the extent of duplication. For test 1, http://www.auburn.edu/ and http://131.204.2.37/ are taken as a sample. Because the latter is the IP address of the former, they are actually the same host. So, we expect the system to tell us that they are at least of level-3 mirroring hosts.

First we ran the spider twice to collect URLs. Once http://www.auburn.edu/ was used as the seed and the other time http://131.204.2.37/ as the seed. After running the spider, 200 URLs were collected from 10 hosts. The URL distribution information is given in Table 3. At first, a set of host feature tuples were generated (Table 4). After applying the algorithm described in Chapter 4, only one host pair is left, which is what we expected: http://www.auburn.edu/ and http://131.204.2.37/.
<table>
<thead>
<tr>
<th>Hosts</th>
<th>Number of URLs</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.alumni.auburn.edu/">http://www.alumni.auburn.edu/</a></td>
<td>27</td>
</tr>
<tr>
<td><a href="http://www.auburn.edu/">http://www.auburn.edu/</a></td>
<td>114</td>
</tr>
<tr>
<td><a href="http://www.aum.edu/">http://www.aum.edu/</a></td>
<td>1</td>
</tr>
<tr>
<td><a href="http://www.eng.auburn.edu/">http://www.eng.auburn.edu/</a></td>
<td>6</td>
</tr>
<tr>
<td><a href="http://www.forestry.auburn.edu/">http://www.forestry.auburn.edu/</a></td>
<td>2</td>
</tr>
<tr>
<td><a href="http://www.grad.auburn.edu/">http://www.grad.auburn.edu/</a></td>
<td>2</td>
</tr>
<tr>
<td><a href="http://www.humsci.auburn.edu/">http://www.humsci.auburn.edu/</a></td>
<td>2</td>
</tr>
<tr>
<td><a href="http://www.univrel.auburn.edu/">http://www.univrel.auburn.edu/</a></td>
<td>22</td>
</tr>
<tr>
<td><a href="http://www.vetmed.auburn.edu/">http://www.vetmed.auburn.edu/</a></td>
<td>2</td>
</tr>
<tr>
<td><a href="http://131.204.2.37/">http://131.204.2.37/</a></td>
<td>86</td>
</tr>
</tbody>
</table>

Table 3: URL distribution from test 1
<table>
<thead>
<tr>
<th>Feature</th>
<th>Word Bi-grams</th>
<th>Position</th>
<th>Host</th>
</tr>
</thead>
<tbody>
<tr>
<td>Admissio</td>
<td>index</td>
<td>4</td>
<td><a href="http://131.204.2.37/">http://131.204.2.37/</a></td>
</tr>
<tr>
<td>Admissio</td>
<td>index</td>
<td>4</td>
<td><a href="http://www.auburn.edu/">http://www.auburn.edu/</a></td>
</tr>
<tr>
<td>Affairs</td>
<td>admissio</td>
<td>3</td>
<td><a href="http://131.204.2.37/">http://131.204.2.37/</a></td>
</tr>
<tr>
<td>Affairs</td>
<td>admissio</td>
<td>3</td>
<td><a href="http://www.auburn.edu/">http://www.auburn.edu/</a></td>
</tr>
<tr>
<td>Affairs</td>
<td>registrar</td>
<td>3</td>
<td><a href="http://131.204.2.37/">http://131.204.2.37/</a></td>
</tr>
<tr>
<td>Affairs</td>
<td>registrar</td>
<td>3</td>
<td><a href="http://www.auburn.edu/">http://www.auburn.edu/</a></td>
</tr>
<tr>
<td>Info</td>
<td>Au</td>
<td>1</td>
<td><a href="http://131.204.2.37/">http://131.204.2.37/</a></td>
</tr>
<tr>
<td>Info</td>
<td>Au</td>
<td>1</td>
<td><a href="http://www.auburn.edu/">http://www.auburn.edu/</a></td>
</tr>
<tr>
<td>Info</td>
<td>student</td>
<td>1</td>
<td><a href="http://131.204.2.37/">http://131.204.2.37/</a></td>
</tr>
<tr>
<td>Info</td>
<td>student</td>
<td>1</td>
<td><a href="http://www.auburn.edu/">http://www.auburn.edu/</a></td>
</tr>
<tr>
<td>Outreach</td>
<td>DI</td>
<td>0</td>
<td><a href="http://131.204.2.37/">http://131.204.2.37/</a></td>
</tr>
<tr>
<td>Outreach</td>
<td>DI</td>
<td>0</td>
<td><a href="http://www.auburn.edu/">http://www.auburn.edu/</a></td>
</tr>
<tr>
<td>Student</td>
<td>Affairs</td>
<td>2</td>
<td><a href="http://131.204.2.37/">http://131.204.2.37/</a></td>
</tr>
<tr>
<td>Student</td>
<td>Affairs</td>
<td>2</td>
<td><a href="http://www.auburn.edu/">http://www.auburn.edu/</a></td>
</tr>
<tr>
<td>Student</td>
<td>Info</td>
<td>0</td>
<td><a href="http://131.204.2.37/">http://131.204.2.37/</a></td>
</tr>
<tr>
<td>Student</td>
<td>Info</td>
<td>0</td>
<td><a href="http://www.auburn.edu/">http://www.auburn.edu/</a></td>
</tr>
</tbody>
</table>

Table 4: Features generated in test 1 (sift value = 5)
The structural similarity score was calculated from the set of feature-host tuples. This score measures the similarity of directory structure of two hosts. It varies with the sift value and number of URLs (Figure 11). There is no statistically significant correlation between the sift value and the similarity score (R= -0.452, p>0.05). Basically, selecting any number between 1 and 10 is acceptable. Selecting 1 will take all features into consideration, which loses the advantage of sifting. And large sift value may cause no features considered. From Figure 11 we can see there are no similarity scores for sift value 14, 16, 17, 21, 22, 23, and 25.

The minimum number of URLs was 50 at the beginning. Even if it was set to 5, still only one pair was generated. It indicated that the system is able to tell us the right candidates from noise.

Figure 11: Similarity scores with different sift values
The validation and classification processing randomly chose 9 URLs from each of http://www.auburn.edu/ and http://131.204.2.37/ (Table 5). These paths are used to retrieve documents from both hosts, namely source host and target host. Then the resemblances of the two documents are compared. There are totally 19 comparisons plus the root page comparison.

<table>
<thead>
<tr>
<th>Paths from <a href="http://131.204.2.37/">http://131.204.2.37/</a></th>
<th>Paths from <a href="http://www.auburn.edu/">http://www.auburn.edu/</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>/outreach/administration.html</td>
<td>/au_index.html</td>
</tr>
<tr>
<td>/student_info/circle/</td>
<td>/main/calendar.html</td>
</tr>
<tr>
<td>/administration/trustees/page1.htm</td>
<td>/cgs/welcome.html</td>
</tr>
<tr>
<td>/outreach/update1299</td>
<td>/duc/policies/network_policy.html</td>
</tr>
<tr>
<td>/duc/policies/network_policy.html</td>
<td>/honors</td>
</tr>
<tr>
<td>/outreach/events</td>
<td>/outreach/dl</td>
</tr>
<tr>
<td>/student_info/student_affairs/registrar/au_registrar_tran.htm</td>
<td>/research/vpr/security/index.html</td>
</tr>
<tr>
<td>/outreach/events/web67.html</td>
<td>~upc/calendar.html</td>
</tr>
<tr>
<td>/co-op/</td>
<td>/research/vpr/biogrants/index.html</td>
</tr>
</tbody>
</table>
Figure 12: Two Auburn University root pages at different time
In this test, except that the root page comparison is Full Similarity (FS), all the other 18 tests are Fingerprint Match (FM). Based on our classification criteria, http://www.auburn.edu/ and http://131.204.2.37/ are level-2 mirror hosts. This means they have same structure and equivalent content. (Sometimes, even High Similarity (HS) can be obtained because of dynamic SSI.) This is because the root page of Auburn University has server side scripts to randomly update the 4 pictures on that page (Figure 12). With different picture names, even two root pages at different time will not be byte wise identical. As a matter of fact, level-1 mirroring is very rare because almost every web site has some pages with dynamic content.

5.4.2 A Level-4 Mirror Hosts Testing

The system is able to detect partial mirror hosts, too. We use “Chip directory” to test this capability. “Chip directory” is a popular directory that offers information about computer chip products, chip manufacturers, and lots of other useful information. It has mirror sites all over the world (Table 6). These mirror sites are not comprehensive. They are under a directory named chipdir, i.e. only part of the host is mirrored. Most of the chipdir directories are first-position directories next to the host names, e.g., http://www.islenet.org/chipdir/. Some of the chipdir are second-position directories, e.g., http://www.ntua.gr/electronics/chipdir/. Using the mirror detecting algorithm, we are able to discover that hosts with first-position chipdir are mirrored hosts to one another in the level 4 category. The same results were found for hosts with second-position chipdir.
<table>
<thead>
<tr>
<th>Continent</th>
<th>Country/State</th>
<th>City</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>America</td>
<td>Argentina</td>
<td>La Plata</td>
<td><a href="http://www.barcala.ing.unlp.edu.ar/chipdir/">http://www.barcala.ing.unlp.edu.ar/chipdir/</a></td>
</tr>
<tr>
<td></td>
<td>USA/California</td>
<td>San Francisco</td>
<td><a href="http://www.hitex.com/chipdir/">http://www.hitex.com/chipdir/</a></td>
</tr>
<tr>
<td></td>
<td>USA/California</td>
<td>San Jose</td>
<td><a href="http://www.embeddedlinks.com/chipdir/">http://www.embeddedlinks.com/chipdir/</a></td>
</tr>
<tr>
<td></td>
<td>USA/Georgia</td>
<td>Atlanta</td>
<td><a href="http://www.islenet.org/chipdir/">http://www.islenet.org/chipdir/</a></td>
</tr>
<tr>
<td></td>
<td>USA/Indiana</td>
<td>Indianapolis</td>
<td><a href="http://www.i-r.net/chipdir/">http://www.i-r.net/chipdir/</a></td>
</tr>
<tr>
<td></td>
<td>USA/Maine</td>
<td>Rockport 1</td>
<td><a href="http://www.avocetsystems.com/chipdir/">http://www.avocetsystems.com/chipdir/</a></td>
</tr>
<tr>
<td></td>
<td>USA/Maine</td>
<td>Rockport 2</td>
<td>(same)</td>
</tr>
<tr>
<td></td>
<td>USA/Maine</td>
<td>Rockport 3</td>
<td>(same)</td>
</tr>
<tr>
<td></td>
<td>USA/Michigan</td>
<td>Okemos</td>
<td><a href="http://www.twinight.org/chipdir/">http://www.twinight.org/chipdir/</a></td>
</tr>
<tr>
<td></td>
<td>USA/Pennsylvania</td>
<td>Pittsburgh</td>
<td><a href="http://www.rabidpenguin.org/chipdir/">http://www.rabidpenguin.org/chipdir/</a></td>
</tr>
<tr>
<td></td>
<td>USA/Texas</td>
<td>Austin</td>
<td><a href="http://www.chipdir.com/chipdir/">http://www.chipdir.com/chipdir/</a></td>
</tr>
<tr>
<td></td>
<td>USA/Washington</td>
<td>Seattle</td>
<td><a href="http://www.amc.com/chipdir/">http://www.amc.com/chipdir/</a></td>
</tr>
<tr>
<td>Asia</td>
<td>Indonesia</td>
<td>Surabaya</td>
<td><a href="http://chipdir.stts.edu/">http://chipdir.stts.edu/</a></td>
</tr>
<tr>
<td></td>
<td>Korea</td>
<td>Seoul</td>
<td><a href="http://icat.snu.ac.kr/chipdir/">http://icat.snu.ac.kr/chipdir/</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sydney</td>
<td><a href="http://www.ideal.net.au/chipdir/">http://www.ideal.net.au/chipdir/</a></td>
</tr>
<tr>
<td>Europe</td>
<td>Austria</td>
<td>St. Pölten</td>
<td><a href="http://mirror.nol.at/chipdir/">http://mirror.nol.at/chipdir/</a></td>
</tr>
<tr>
<td></td>
<td>Germany</td>
<td>Kaiserslautern</td>
<td><a href="http://www.fh-kl.de/~rscherer/chipdir/">http://www.fh-kl.de/~rscherer/chipdir/</a></td>
</tr>
<tr>
<td></td>
<td>Greece</td>
<td>Athena</td>
<td><a href="http://www.ntua.gr/electronics/chipdir/">http://www.ntua.gr/electronics/chipdir/</a></td>
</tr>
<tr>
<td></td>
<td>Italy</td>
<td>Napoli</td>
<td><a href="http://ftp.unina.it/pub/chipdir/">http://ftp.unina.it/pub/chipdir/</a></td>
</tr>
<tr>
<td></td>
<td>Netherlands</td>
<td>Amsterdam 1</td>
<td><a href="http://www.xs4all.nl/~ganswijk/chipdir/">http://www.xs4all.nl/~ganswijk/chipdir/</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Amsterdam 2</td>
<td><a href="http://www.chipdir.nl/">http://www.chipdir.nl/</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Enschede</td>
<td><a href="http://margo.student.utwente.nl/stefan/chipdir/">http://margo.student.utwente.nl/stefan/chipdir/</a></td>
</tr>
<tr>
<td></td>
<td>Portugal</td>
<td>Lisboa</td>
<td><a href="http://junitc.ist.utl.pt/chipdir/">http://junitc.ist.utl.pt/chipdir/</a></td>
</tr>
<tr>
<td></td>
<td>Russia</td>
<td>Moscow</td>
<td><a href="http://www.chipinfo.ru/chipdir/">http://www.chipinfo.ru/chipdir/</a></td>
</tr>
<tr>
<td></td>
<td>Spain</td>
<td>Barcelona</td>
<td><a href="http://www.fer.nu/chipdir/">http://www.fer.nu/chipdir/</a></td>
</tr>
<tr>
<td></td>
<td>Sweden</td>
<td>Stockholm</td>
<td><a href="http://www.zettweb.com/semiconductors/chipdir/">http://www.zettweb.com/semiconductors/chipdir/</a></td>
</tr>
</tbody>
</table>

Table 6: Mirror sites of chip directory
To demonstrate the detecting steps, we randomly picked up two hosts from Table 9: http://www.amc.com/chipdir/ and http://www.chipdir.com/chipdir/. Totally 186 URLs were collected for this test, 76 from http://www.amc.com/ and 74 from http://www.chipdir.com/ (Table 7). Using 5 as a sift value, we got a set of feature host tuples (Table 8). The candidate mirror generator generated only one pair of candidates: http://www.amc.com/ and http://www.chipdir.com/. The structural similarity score between them is 3.16.

<table>
<thead>
<tr>
<th>Host</th>
<th>Number of URLs</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.amc.com/">http://www.amc.com/</a></td>
<td>76</td>
</tr>
<tr>
<td><a href="http://www.chipdir.com/">http://www.chipdir.com/</a></td>
<td>74</td>
</tr>
<tr>
<td>Others</td>
<td>36</td>
</tr>
<tr>
<td><strong>Total:</strong></td>
<td><strong>186</strong></td>
</tr>
</tbody>
</table>

Table 7: Chipdir URLs
Table 8: Features generated (sift value = 5)

<table>
<thead>
<tr>
<th>Feature</th>
<th>Word Bi-gram</th>
<th>Position</th>
<th>Host</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chipdir</td>
<td>C</td>
<td>0</td>
<td><a href="http://www.amc.com/">http://www.amc.com/</a></td>
</tr>
<tr>
<td>Chipdir</td>
<td>C</td>
<td>0</td>
<td><a href="http://www.chipdir.com/">http://www.chipdir.com/</a></td>
</tr>
<tr>
<td>Chipdir</td>
<td>Mirror</td>
<td>0</td>
<td><a href="http://www.amc.com/">http://www.amc.com/</a></td>
</tr>
<tr>
<td>Chipdir</td>
<td>Mirror</td>
<td>0</td>
<td><a href="http://www.chipdir.com/">http://www.chipdir.com/</a></td>
</tr>
<tr>
<td>Chipdir</td>
<td>N</td>
<td>0</td>
<td><a href="http://www.amc.com/">http://www.amc.com/</a></td>
</tr>
<tr>
<td>Chipdir</td>
<td>N</td>
<td>0</td>
<td><a href="http://www.chipdir.com/">http://www.chipdir.com/</a></td>
</tr>
<tr>
<td>Htm</td>
<td>Mirror</td>
<td>2</td>
<td><a href="http://www.amc.com/">http://www.amc.com/</a></td>
</tr>
<tr>
<td>Htm</td>
<td>Mirror</td>
<td>2</td>
<td><a href="http://www.chipdir.com/">http://www.chipdir.com/</a></td>
</tr>
<tr>
<td>Htm</td>
<td>MI</td>
<td>2</td>
<td><a href="http://www.amc.com/">http://www.amc.com/</a></td>
</tr>
<tr>
<td>Htm</td>
<td>MI</td>
<td>2</td>
<td><a href="http://www.chipdir.com/">http://www.chipdir.com/</a></td>
</tr>
<tr>
<td>Htm</td>
<td>N</td>
<td>2</td>
<td><a href="http://www.amc.com/">http://www.amc.com/</a></td>
</tr>
<tr>
<td>Htm</td>
<td>N</td>
<td>2</td>
<td><a href="http://www.chipdir.com/">http://www.chipdir.com/</a></td>
</tr>
<tr>
<td>Htm</td>
<td>Soviet</td>
<td>2</td>
<td><a href="http://www.amc.com/">http://www.amc.com/</a></td>
</tr>
<tr>
<td>Htm</td>
<td>Soviet</td>
<td>2</td>
<td><a href="http://www.chipdir.com/">http://www.chipdir.com/</a></td>
</tr>
<tr>
<td>Htm</td>
<td>Xicor</td>
<td>2</td>
<td><a href="http://www.amc.com/">http://www.amc.com/</a></td>
</tr>
<tr>
<td>Mirror</td>
<td>Htm</td>
<td>1</td>
<td><a href="http://www.amc.com/">http://www.amc.com/</a></td>
</tr>
<tr>
<td>Mirror</td>
<td>Htm</td>
<td>1</td>
<td><a href="http://www.chipdir.com/">http://www.chipdir.com/</a></td>
</tr>
<tr>
<td>MI</td>
<td>Chipdir</td>
<td>3</td>
<td><a href="http://www.amc.com/">http://www.amc.com/</a></td>
</tr>
<tr>
<td>MI</td>
<td>Chipdir</td>
<td>3</td>
<td><a href="http://www.chipdir.com/">http://www.chipdir.com/</a></td>
</tr>
<tr>
<td>Soviet</td>
<td>Index</td>
<td>3</td>
<td><a href="http://www.amc.com/">http://www.amc.com/</a></td>
</tr>
<tr>
<td>Soviet</td>
<td>Index</td>
<td>3</td>
<td><a href="http://www.chipdir.com/">http://www.chipdir.com/</a></td>
</tr>
<tr>
<td>Xicor</td>
<td>Index</td>
<td>3</td>
<td><a href="http://www.amc.com/">http://www.amc.com/</a></td>
</tr>
<tr>
<td>Xicor</td>
<td>Index</td>
<td>3</td>
<td><a href="http://www.chipdir.com/">http://www.chipdir.com/</a></td>
</tr>
</tbody>
</table>

To verify and classify this candidate mirror host pair, 18 paths were randomly selected to conduct cross validation tests (Table 9). All 18 tests showed Fingerprint Match (FM). But the root pages of the two hosts are so different and the resemblance between them is No Similarity (NS). According to the classification criteria, their mirroring extent is in level 4. This result validates the partial duplication of the two hosts.
<table>
<thead>
<tr>
<th>URL</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>/chipdir/mirror.htm?n/8.htm</td>
<td>/chipdir/n/index.htm</td>
</tr>
<tr>
<td>/chipdir/mirror.htm?giicm/index.htm</td>
<td>/chipdir/mirror.htm?xicor/index.htm</td>
</tr>
<tr>
<td>/chipdir/giicm/index.htm</td>
<td>/chipdir/xicor/index.htm</td>
</tr>
<tr>
<td>/chipdir/n/8.htm</td>
<td>/chipdir/mirror.htm?n/index.htm</td>
</tr>
<tr>
<td>/chipdir/mirror.htm?ml/chipdir.htm</td>
<td>/chipdir/mirror.htm?prefix/index.htm</td>
</tr>
<tr>
<td>/chipdir/pin/index.htm</td>
<td>/chipdir/pin/index.htm</td>
</tr>
<tr>
<td>/chipdir/mirror.htm?xicor/index.htm</td>
<td>/chipdir/mirror.htm?s/3volt.htm</td>
</tr>
<tr>
<td>/chipdir/mirror.htm?s/3volt.htm</td>
<td>/chipdir/mirror.htm?moredata.htm</td>
</tr>
<tr>
<td>/chipdir/mirror.htm?n/0.htm</td>
<td>/chipdir/cable/index.htm</td>
</tr>
</tbody>
</table>


However, we are not able to detect the mirroring relationship between hosts with first positional chipdir and those with second positional chipdir. This is because the algorithm relies on syntactic analysis of URL strings. Hosts with the same content but different path cannot be detected.
6 Conclusion and Future Work

This chapter concludes the project and provides some future research topics on mirror hosts detecting.

6.1 Conclusions

Previous studies on the WWW have identified mirroring as a major cause of duplication on the web but have not analyzed the mirroring issue in depth. This project introduced an algorithm to measure systematic replication among hosts. A web based mirror detecting application was created that includes a web spider and mirror detection processes. A JSWDK server was installed to support Java Servlet funcionality. The experimental results suggested that this application is capable of detecting total mirroring and partial mirroring, even with small number of URLs. This technique is efficient because it operates syntactically on URL strings and requires fetching only about 20 pages per prospective mirrored host for confirmatory content analysis. It could be used for detecting mirrors in web proxies and search engines in order to reduce redundant fetching and to improve caching behavior and reliability.

6.2 Future Work

Mirroring information is useful in the implementation of smart caching proxies and efficient crawlers. Levels 1 to 3, according to our classification, indicate pairs of hosts that can be used interchangeably. A proxy that maintains such a list of mirrors can serve a cached page from any of the mirrors of a given host, provided that the path is the same. A smart proxy can also try and compensate for broken links or server failure by
transparently checking if the page is available on a mirror site. A crawler that tries to
cover as much as the Web as possible in the shortest possible time can use mirroring
information to avoid redundant crawling over mirrored parts of the Web. Also, in the
interests of load balancing or in the interests of speed it can selectively download a path
from an equivalent host.

Several recent ranking algorithms that rely on connectivity treat hyperlinks within
a host differently from hyperlinks across hosts. Hyperlinks in the latter case are
considered indicative of quality, since they suggest a form of endorsement across
organizations. Mirroring information has the potential to make these algorithms more
robust and accurate, because it can identify organizational bounders that span many hosts.

Broken links on the WWW are a common problem. The use of publishing tools
and inter-server communication protocols has been advocated to preserve referential
integrity when links change. Until such a protocol is adopted globally, and legacy servers
are retired, we will continue to see Page Not Found errors. When a broken link refers to a
known replicated host, mirroring information can be used to fetch an equivalent copy of
the page.
REFERENCES

   

   http://www.oclc.org/oclc/research/publications/review97/oneill/o'neilla%r980213.htm


