Topic-oriented Collaborative Web Crawling

by

Chiasen (Charles) Chung

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2001

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

A *web crawler* is a program that "walks" the Web to gather web resources. In order to scale to the ever-increasing Web, multiple crawling agents may be deployed in a distributed fashion to retrieve web data co-operatively. A common approach is to divide the Web into many partitions with an agent assigned to crawl within each one. If an agent obtains a web resource that is not from its partition, the resource will be transferred to the rightful owner.

This thesis proposes a novel approach to distributed web data gathering by partitioning the Web into topics. The proposed approach employs multiple focused crawlers to retrieve pages from various topics. When a crawler retrieves a page of another topic, it transfers the page to the appropriate crawler. This approach is known as *topic-oriented collaborative web crawling.*

An implementation of the system was built and experimentally evaluated. In order to identify the topic of a web page, a topic classifier was incorporated into the crawling system. As the classifier categorizes only English pages, a language identifier was also introduced to distinguish English pages from the non-English ones. From the experimental results, we found that redundance retrieval was low and that a resource, retrieved by an agent, is six times more likely to be retained than a system that uses conventional hashing approach. These numbers were viewed as strong indications that *topic-oriented collaborative web crawling system* is a viable approach to web data gathering.

Acknowledgement

First in the list, I would like to thank my supervisor Charlie Clarke, so I will do exactly that. Thank you, Charlie. It is my pleasure to be under your supervision. The completion of this thesis would not be possible if not for your guidance and constructive suggestions. The pain you went through when reading my drafts is also appreciated.

This research would not be carried out as smoothly as it had if not for the people in Programming Languages Group (PLG), particularly Greg McLearn for keeping the Canolas and Flaxes actively alive.

Special thanks to Parbhakar Ragde for taking me into the department and to all the wonderful people I met in Waterloo who have helped in ways you may never know.

My heartfelt gratitude to Leilei Zeng for waiting patiently on countless nights and to be always there when I needed someone to talk to, both in the aspect of research and not. Most of all, I am grateful to my family for being with me all these time. Though you are half a world away, your presence have been felt and deeply appreciated.

Contents

1 Introduction **1**

1.1 Motivation 1

1.2 Thesis' objective 2

1.3 Organization of Thesis 3

2 Background and related work **5**

2.1 Hypertext Transfer Protocol (HTTP) 6

2.2 Hypertext Markup Language (HTML) 10

2.2.1 Parsing HTML page 10

2.2.2 URL extraction 13

2.2.3 Language support 14

2.3 Searching the web 15

2.3.1 Search engines 16

2.3.2 Query Expansion 19

2.4 Link analysis in a hypertext environment 20

2.5 Web crawling 23

2.5.1 Web crawler anatomy 23

2.5.2 Focused crawling 26
2.5.3 Distributed crawling 27

3 Distributed web crawling **29**

3.1 Non-centralized distributed crawling 30
3.1.1 Web partitioning 31
3.2 Multitext crawler 36
3.2.1 Design of the crawler 36
3.2.2 Tasks of child processes 37
3.2.3 System considerations 39
3.3 The topic-oriented crawling system 41
3.3.1 Topic-oriented partitioning 42
3.3.2 Architecture of C4 43

4 Classification of web pages **48**

4.1 Classification background 49
4.1.1 Corpus 50
4.1.2 Term representation 51
4.1.3 Document indexing 53
4.1.4 Classification technique 55
4.1.5 Feature engineering 58
4.1.6 Document clustering 60
4.2 C4 Topic Classifier 61
4.2.1 Training and testing data 61
4.3 Classification experiments 64

4.3.1 Classification Methods Comparison 65
4.3.2 Hyperlink Relationship 66
4.3.3 Classification throughput 67
4.3.4 Discussion 68
4.4 Language Identification 68
4.4.1 Classical language identification techiques. 70
4.4.2 Language identification on the Web 71
4.4.3 Design and performance 71
4.5 Components of the C4 Classifier 74

5 Experimental results **76**
5.1 Experimental setup 76
5.2 Results 78
5.2.1 General retrieval results 78
5.2.2 Topic classification 79
5.2.3 Duplicate retrieval by crawlers 80
5.2.4 Transfer of pages between English and foreign crawlers . . . 81
5.3 Discussion 83
5.3.1 Crawlers' retrieval rate 83
5.3.2 URL duplication 83
5.3.3 The influence of unclassifiable pages 84
5.3.4 Performance evaluation of the classifier 85
5.3.5 The feasibility of topic-oriented partitioning 87

6 Conclusion **88**

6.1 Future work 89

Bibliography **91**

A URLs distribution to crawlers **101**

B Data distribution to crawlers **107**

C Percentage of distribution to crawlers **113**

List of Tables

2.1 Description of different sets of HTTP status code 9

2.2 Elements that supports *href* and *src* attribute. 13

2.3 Some examples of entity characters that are commonly found in HTML pages. 15

4.1 Top categories in ODP chosen to be used in the English topic classifier. 62

4.2 Size of training and testing data gathered for each category. 64

5.1 The number of URLs that were identified as unclassifiable during the pre-processing stage of the topic-classifer. The values in brackets indicate the corresponding amount of retrieved data (in GB). 81

List of Figures

2.1 An example of a HTTP respond packet that indicates a successfully request 8

2.2 An example of a HTTP respond packet that indicates a redirected request 10

2.3 An example of an embedded script in a HTML page. The line embedded in the HTML comment is a program encoded in JavaScript. 12

2.4 An example of attaching an external script file in a HTML page by using the "src" attribute of the "SCRIPT" element. 12

2.5 A conceptual example of a link relationship between hubs and authorities. 21

3.1 Centralized servers. 31

3.2 Hierarchical organisation of centralised servers. 32

3.3 High-level view of the design of Multitext crawler. 40

3.4 Resolve process diagram. 41

3.5 C4 classifier and gofer processes for crawler #i. 44

3.6 Pre-processing stage of C4 Classifier, in which unclassifiable web pages are identified and are assigned with categories. 47

4.1 Performance of classifiers on different categories. 66
4.2 Likelihood of a link to be classified under the same category as its parent page. 67
4.3 Distribution of $\Pr(\delta)$ across different ranges of cutoff K 73
4.4 Categorization of classifiable web pages by C4 Classifier. 74

5.1 Duration of the crawlers in execution mode. 78
5.2 Number of URLs retrieved by each crawler. 79
5.3 Amount of web data retrieved by each crawler. 80
5.4 Percentage of classifiable URLs assigned to the same category as its parent page. This result is compared against the theoretical result of a hashing approach, which is constant at 6.25% across all categories. 82
5.5 Amount of duplicate pages retrieved by multiple crawlers. 82
5.6 The rate of retrieval of each crawler, with an averaged of 13.2 KB/sec. 83
5.7 Ratio of classifiable pages against the unclassifiable ones. 84

A.1 Distribution of URLS from *Adult* crawler to others. 101
A.2 Distribution of URLS from *Arts* crawler to others. 101
A.3 Distribution of URLS from *Business* crawler to others. 102
A.4 Distribution of URLS from *Computers* crawler to others. 102
A.5 Distribution of URLS from *Games* crawler to others. 102
A.6 Distribution of URLS from *Health* crawler to others. 103

A.7 Distribution of URLS from *Home* crawler to others. 103
A.8 Distribution of URLS from *Kids/Teens* crawler to others. 103
A.9 Distribution of URLS from *News* crawler to others. 104
A.10 Distribution of URLS from *Recreation* crawler to others. 104
A.11 Distribution of URLS from *Reference* crawler to others. 104
A.12 Distribution of URLS from *Science* crawler to others. 105
A.13 Distribution of URLS from *Shopping* crawler to others. 105
A.14 Distribution of URLS from *Society* crawler to others. 105
A.15 Distribution of URLS from *Sports* crawler to others. 106
A.16 Distribution of URLS from *World* crawler to others. 106

B.1 Amount of data distributed from *Adult* crawler to others. 107
B.2 Amount of data distributed from *Artx* crawler to others. 107
B.3 Amount of data distributed from *Business* crawler to others. 108
B.4 Amount of data distributed from *Computers* crawler to others. . . . 108
B.5 Amount of data distributed from *Games* crawler to others. 108
B.6 Amount of data distributed from *Health* crawler to others. 109
B.7 Amount of data distributed from *Home* crawler to others. 109
B.8 Amount of data distributed from *Kids/Teens* crawler to others. . . 109
B.9 Amount of data distributed from *News* crawler to others. 110
B.10 Amount of data distributed from *Recreation* crawler to others. . . . 110
B.11 Amount of data distributed from *Reference* crawler to others. . . . 110
B.12 Amount of data distributed from *Science* crawler to others. 111
B.13 Amount of data distributed from *Shopping* crawler to others. 111

B.14 Amount of data distributed from *Society* crawler to others. 111
B.15 Amount of data distributed from *Sports* crawler to others. 112
B.16 Amount of data distributed from *World* crawler to others. 112

C.1 Percentage of distribution (of URLs and Web data) from *Adult* crawler to others. 113
C.2 Percentage of distribution (of URLs and Web data) from *Arts* crawler to others. 114
C.3 Percentage of distribution (of URLs and Web data) from *Business* crawler to others. 114
C.4 Percentage of distribution (of URLs and Web data) from *Computers* crawler to others. 114
C.5 Percentage of distribution (of URLs and Web data) from *Games* crawler to others. 115
C.6 Percentage of distribution (of URLs and Web data) from *Health* crawler to others. 115
C.7 Percentage of distribution (of URLs and Web data) from *Home* crawler to others. 115
C.8 Percentage of distribution (of URLs and Web data) from *Kids/Teens* crawler to others. 116
C.9 Percentage of distribution (of URLs and Web data) from *News* crawler to others. 116
C.10 Percentage of distribution (of URLs and Web data) from *Recreation* crawler to others. 116

C.11 Percentage of distribution (of URLs and Web data) from *Reference* crawler to others. 117
C.12 Percentage of distribution (of URLs and Web data) from *Science* crawler to others. 117
C.13 Percentage of distribution (of URLs and Web data) from *Shopping* crawler to others. 117
C.14 Percentage of distribution (of URLs and Web data) from *Society* crawler to others. 118
C.15 Percentage of distribution (of URLs and Web data) from *Sports* crawler to others. 118
C.16 Percentage of distribution (of URLs and Web data) from *World* crawler to others. 118

Chapter 1

Introduction

A *web crawler* is a program that "walks" the Web to gather web resources. Given a Uniform Resource Locator (URL), the crawler retrieves the corresponding web page. The links are then extracted from this page and stored into a queue. The next URL is removed from the queue and the retrieval process is repeated. The retrieved pages are retained for future processing. There are various application of web crawling, such as to retrieve pages with specific content [7] or to create an indexable database of web pages for use by a search engine [1].

1.1 Motivation

The Web has been growing at a dramatic rate. As the amount of web data increases, retrieving a good representative portion of the Web is becoming an increasing challenge for a single crawler. Therefore multiple crawling agents (web crawlers) are often deployed in a distributed fashion to retrieve web data co-operatively. A com-

mon approach is to divide the Web into partitions with an agent assigned to crawl each one. When an agent obtains a web resource that does not belong to its partition, the resource will be transferred to the rightful owner. As overheads for transferring of data are costly, it is inefficient if external resources are retrieved frequently by the agents.

1.2 Thesis' objective

Web pages of similar interest are often linked to each others, hence a crawler that is assigned to retrieve pages of a particular topic tends to retrieve pages from that topic. Based on this observation, a novel approach to web data gathering is presented in this thesis. An agent is assigned to crawl web pages of a specific topic. When it retrieves a web page, it examines the content to determine the topic to which the page shall be assigned. If the page is assigned to the agent's topic, it will be retained. Otherwise, it will be routed to the appropriate external agent. This is known as *topic-oriented partitioning* approach.

In conventional web crawling systems, the transferred resources do not contribute to the crawlers' performance. However, the crawlers in a topic-partitioned system collaborated on the retrieval by continuously supplying relevant web pages to each other. These pages link to more pages of the same topic, thus reinforcing the inclination of each crawler to retrieve pages from its partition and reduce data transfer between the crawlers.

In order to identify the topic of a web page, a topic classifier is used. This thesis identifies a classification technique that is suitable for use by the crawling system.

Topic classifiers are language-dependent. In other words, if a topic classifier is trained using a Chinese corpus, classification on non-Chinese pages will produce meaningless results. Therefore the natural language of a document needs to be identified before it can be classified into a topic. As this thesis focuses on classifying English pages, a simple and fast language identifier that distinguishes English pages from non-English ones is introduced. When a page is retrieved, the identifier first determines if it is an English page. If it is, it will be categorized by the topic classifier into a set of topics. Otherwise, it is placed in a separate "non-English" category. To support the thesis, an implementation of the topic-oriented collaborative web crawling system named *C4* was built.

1.3 Organization of Thesis

The following describes the organization of the thesis:

Chapter 2 provides the background to web crawling. HTTP and HTML are reviewed with a focus on the aspects of the protocol that are important for web crawler implementation. Crawlers are widely used by search engines, therefore a section is dedicated to searching on the Web. The graph structure of the Web is also introduced here. Finally various design concerns and implementation issues of web crawlers are explored.

Chapter 3 discusses distributed web crawling. It begins by differentiating centralized and non-centralized methods for distributed web crawling system. Various web partitioning methods are introduced. It then discusses the architecture of the Multitext crawler. Finally, *C4*, the collaborative crawling system, is presented.

In chapter 4, we look into classification of web pages. The background to classification is first studied. A topic classifier and a language identifier that are suitable to be used in *C4* are then identified.

Chapter 5 presents the results of a crawling experiment carried out with *C4* over a period of 28 days. This experiment focuses on the retrieval rate of the crawling system and data redundancy among the crawlers.

The final chapter provides a summary of the thesis. It also analyses the overall performance of *C4* and the feasibility of the topic-oriented partitioning approach. It then concludes the thesis with a list of possible future work.

Chapter 2

Background and related work

This thesis proposes a novel approach to distributed crawling system. Before the approach is detailed, the background pertaining to web crawling is presented in this chapter. Web crawling involves the retrieval of web resources such as HTML pages and images. The first section covers the protocol for transferring web resources over the Internet with a focus on the aspects of the protocol that are important for web crawler implementation. The second section looks into the issues of parsing HTML pages to extract URLs that are needed for crawlers to traverse the Web. As the Web contains documents of multiple languages, the means to identify the natural language of these documents are also outlined in these two sections. Section three explores the issues of searching the Web, follows by an introduction to link analysis on the Web. The final section discusses the anatomy of a web crawler.

2.1 Hypertext Transfer Protocol (HTTP)

The Hypertext Transfer Protocol [20] is an application-level protocol for distributed, collaborative, hypermedia information systems running on top of TCP/IP. This protocol has been used by the World Wide Web to transfer resources since 1990. This section provides a quick review of the protocol focusing on the aspects that are important to web crawlers.

URL. Location of web resources, such as web pages and images, are represented by Uniform Resource Locators (URLs) [43]. A URL is a string that makes up of several parts, as follows:

protocol://*host name*:*port number*/*resource path*?*query string*

the protocol, host name and resource path must be present in a URL. In the absence of port number, it defaults to 80. For example, the URL:

`http://plg.uwaterloo.ca:88/~c3chung/search?q=Singapore`

indicates that the resource "/~c3chung/search", with a query "q=Singapore", is obtainable from host "plg.uwaterloo.ca" at port 88 using the HTTP protocol.

HTTP Connection. A HTTP server is a daemon that is commonly bound to port 80. For a client to obtain a web resource, a connection to the remote HTTP server where the resource resides is established. In Unix systems, this connection can be made using a socket, which is an application-level abstraction for data communication between two hosts. After it is connected, the client sends a request

packet. The packet consists of a HTTP request method, the requested file and a HTTP version. For example, to retrieve the web page

http://plg.uwaterloo.ca/~c3chung/index.html

using HTTP version 1.0, a connection to the host plg.uwaterloo.ca is established. The client then writes the following GET request string (which terminates with an empty line) into the connection:

```
GET /~c3chung/index.html HTTP/1.0

```

Based on the request method, the server replies with a respond packet. Every respond packet has a status line, a HTTP header and, possibly, a entity-body content. The status line consists of the message's protocol version and a (success or error) status code. The HTTP header consists of server information and entity meta-information. This header ends with an empty line, after which the start of entity body (if any) follows. After the content body has been sent, the server closes the connection. As every HTTP connection is independent from the previous one, they are considered stateless.

Many types of information can be obtained from the header of response packets, like the size of the requested resource ("Content-Length") and the time when the resource was last modified ("Last-Modified"). The ones that are of importance to web crawlers are the HTTP status code in the first line and the "Content-Type" field. Figure 2.1 shows an example of a response packet. The first line of the packet is the HTTP status line, followed by eight lines of HTTP header. This header is terminated with an empty line, after which the transmission of the requested

resource begins. In this example, the server has responded with a "200" status code and a "Content-Type: text/html" field. These respectively indicate that the resource has been successfully requested and is a HTML web page, which will be transmitted as the entity body.

```
HTTP/1.1 200 OK
Date:  Fri, 15 Jun 2001 01:54:15 GMT
Server:  Apache/1.2.6
Last-Modified:  Thu, 14 Jun 2001 01:38:43 GMT
ETag:  "32b7a6-168a-3b2815a3"
Content-Length:  5770
Accept-Ranges:  bytes
Connection:  close
Content-Type:  text/html

<HTML>
⋮
</HTML>
```

Figure 2.1: An example of a HTTP respond packet that indicates a successfully request

Language support. As the Web is a multi-lingual corpus, documents of various languages are transferred through HTTP. The natural language of the transmitted document is indicated in the "Content-Language" and "Accept-Language" entity-header fields. For example, the appropriate field to indicate a body content that is intended only for a Danish-literate person is:

Content-Language: da

The natural language can be encoded in different character sets. An optional "charset" parameter of the "Content-Type" field is for indicating the character

set used in a web page. For instance, the appropriate field to indicate that the transmitted document is a HTML web page encoded in the "JUNET" encoding of Japanese [46] is: Content-Type: text/html; charset=ISO-2022-JP. In the absence of the "charset" parameter, the language encoding defaults to US-ASCII. US-ASCII is a 7-bit encoding scheme for the English language that uses 8 bits for each character (the most significant bit is always set to zero), hence it is capable of distinguising 128 characters.

1xx	Informational
2xx	Successful
3xx	Redirection
4xx	Client Error
5xx	Server Error

Table 2.1: Description of different sets of HTTP status code

HTTP status code. The 3-digit HTTP status code in the status line of a server's respond packet indicates the validity of a client's request. Table 2.1 shows the different types of status codes. There are two sets of status code that are essential to web crawler implementation: 2XX (successful) and 3XX (redirection) code. The first indicates that the requested resource is available and will be transferred to the client. Pages that contain a 2XX status code are referred to as *valid* pages in this thesis. The second code indicates that the resource is residing in another location (URL). In the case where the resource is a HTML web page, the entity body will be the web page if the request is successful. If the server responded with a redirection, the entity body is usually (though not always) a HTML page that

contains a hypertext link to the URL where the resource resides. Figure 2.1 and 2.2 provide examples of HTTP response packets with these status codes.

```
HTTP/1.1 301 Moved Permanently
Date:  Fri, 15 Jun 2001 03:00:37 GMT
Server:  Apache/1.2.6
Location:  http://plg.uwaterloo.ca/~c3chung/
Connection:  close
Content-Type:  text/html

<HTML>
⋮
</HTML>
```

Figure 2.2: An example of a HTTP respond packet that indicates a redirected request

2.2 Hypertext Markup Language (HTML)

Hypertext Markup Language (HTML) is the universal language used on the World Wide Web. Among many data items that can be embedded into HTML documents are the locations of other web resources, which are represented using URLs. This section focuses on the issues of parsing HTML pages to extract these URLs which are needed by crawlers to traverse the Web.

2.2.1 Parsing HTML page

Before any analysis can be done on a HTML web page, it needs to be parsed. Different parts of a HTML page requires different parsing conditions. A HTML

web page can be broken into three parts: HTML *tags*, *scripts* and *text*.

Tags A HTML tag begins with a "<" and ends with a ">". In the case of a comment tag, it begins and ends with "<!−−" and "−− >" respectively (e.g. <!−−Comment tag−− >, therefore the greater-than ("<") and less-than (">") characters are accepted within it). *Element* refers to the keyword of a tag. Some elements require end tags (which are denoted by a slash (/) before the associated element name) while others do not. For example, <html> and < /html> indicate the start and end tag of an html element. An element may have a set of associated properties call attributes, which are defined within each tag (e.g. <A href="http://www.uwaterloo.ca/index.html"> is a tag of element A with attribute href defined).

Scripts HTML script, such as VBScript and JavaScript, encodes program source code associated with a web page that is executed on the client's machine when the page is down-loaded. There are many applications for scripts, such as multi-media interaction and dynamic web page modification. A script can be attached to a HTML document in two ways. The first way is through *intrinsic events* that are associated with control elements (such as INPUT, SELECT, BUTTON, TEXTAREA, and LABEL) in the form of attributes. For example, the following tag executes a JavaScript function to validate the value when the user leaves the input field:

```
<INPUT NAME="userName" onblur="validUserName(this.value)">
```

These embedded scripts are parsed as HTML tags.

The second way of attaching a script is to use the SCRIPT element. The content of the script can either be embedded into the document between the start and end tags of the element. As not all browsers are script-enabled, scripts are often embedded within a HTML comment tag. This form of embedded scripts are parsed as tags as well (see figure 2.3). Alternatively, the script can be located in an external file which is referred to by the src attribute (see figure 2.4).

```
<HTML>
⋮
<script language="JavaScript">
 <!--
 document.write("Last modified:  " + document.lastModified) ;
 // -->
</script>
⋮
</html>
```

Figure 2.3: An example of an embedded script in a HTML page. The line embedded in the HTML comment is a program encoded in JavaScript.

```
<HTML>
⋮
<SCRIPT language="JavaScript"
 src="http://plg.uwaterloo.ca/~c3chung/Script/code.js">
</SCRIPT>
⋮
</html>
```

Figure 2.4: An example of attaching an external script file in a HTML page by using the "src" attribute of the "SCRIPT" element.

Text Lastly, text is the entire web page without the HTML tags and scripts. In English text, words are often broken into tokens by white space and punctuation.

2.2.2 URL extraction

HTML allows the location of other web resources (URLs) to be specified in web pages. As a crawler parses a HTML web page, it looks for tags that support *src* or *href* attributes (see table 2.2). These attributes contain URLs of other web

Attribute	Elements (tags)
HREF	A, AREA, LINK, BASE
SRC	SCRIPT, INPUT, FRAME, IFRAME, IMG

Table 2.2: Elements that supports *href* and *src* attribute.

resources and are extracted for future crawling. The following is an example of an anchor tag that refers to a web resource:

```
<A href="http://www.uwaterloo.ca/">click here</A>
```

Relative URLs can also be specified by *src* or *href* attributes. For instance, a relative URL

../NeverEnds/this_thesis.html

found in the web page of

http://work.com/My/Tasks/Current/index.html

is resolved into

http://work.com/My/Tasks/NeverEnds/this_thesis.html

A relative URL can also be resolved to a URL of a different host by using the BASE element. This element allows authors to specify a document's base URL explicitly. For example, the following shows the web page of an arbitrary URL:

```
⋮
<BASE href="http://work.com/My/Tasks/Current/">
⋮
Hell breaks loose when this
<A href="../NeverEnds/this_thesis.html">thesis</A> is done
⋮
```

the relative URL

../NeverEnds/this_thesis.html

would also resolve to

http://work.com/My/Tasks/NeverEnds/this_thesis.html.

2.2.3 Language support

Various issues of globalizing HTML have been addressed [69], in order to support multiple languages on the Web. The Lang attribute, a property that is applicable in many elements, specifies the language within the element content. For instance, in response to the following HTML text:

```
<P>As the saying goes: <Q lang="fr">c'est la vie</Q>.</P>
```

a HTML parser should treat the content within the Q element as french.

Beside natural languages, HTML also supports a set of special entity characters that cannot be entered directly using a conventional keyboard. These characters can be represented numerically (either decimal or hexadecimal) or symbolically. For example, the character "©" can be denoted by "©"(decimal) or "©"(symbolic). Escape characters of HTML (or any other ASCII characters) can be represented using numerical entity references which corresponds to their ASCII values (see table 2.2.3).

Escape character	Numerical reference	Symbolic reference
"	&#34;	&quot;
&	&#38;	&amp;
<	&#60;	&lt;
>	&#62;	&gt;

Table 2.3: Some examples of entity characters that are commonly found in HTML pages.

2.3 Searching the web

In December 1997, the Web was estimated to contain at least 320 million indexable web pages [38]. In less than two years, it increased to 800 million pages, containing 6 TB of data on 3 million servers [39]. As the Web grows, a challenge for web search engines is to locate resources, track document changes and find relevant web pages.

2.3.1 Search engines

Search engines, such as Google[1] and Directhit[2], are popular portals into the World Wide Web. Each of them maintains a large database of indexed web pages which are retrieved from the Web by specialized programs known as web crawlers. To search for a document, a user provides a small set of words, known as a query, that is descriptive of the page or site desired. A search engine then uses this query to match against documents in its database. These selected documents are then ranked in order of likely relevancy before their corresponding URLs are presented to the user.

Ranking techniques. Different engines employ different techniques to rank documents. *Content-based* methods observe the degree of similarity between the web pages and the query terms by assigning similarity values based on the occurrence of query terms in the pages [53, 52, 10].

Beside observing the content of a query, web documents can be ranked through *link analysis* methods. *Back-links* of a page are pages that contain a hyperlink to it. In other words, if page A contains a hyperlink to B, then A is a back-link of B. Assuming that pages linked to by many others are considered more important, *back-link counts* keep track of the number of back-links of each page. As the Web is a uni-directional graph, this algorithm requires the retrieval of the entire Web (which is nearly impossible). Hence *back-link* weight is often approximated by considering the pages that a crawler has retrieved so far [9]. An extension to the

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

[2] http://www.directhit.com

HTTP protocol that would greatly improve the feasibility of this ranking technique is to include back-link information, which was proposed by Chakrabarti [6].

The technique of using *back-link counts* has a number of drawbacks. Consider Yahoo's home page (http://www.yahoo.com). It is likely to be referenced (linked to) by many web pages, hence the page is ranked highly using this technique. However an important web page (e.g. http://dir.yahoo.com/News_and_Media/) linked by the Yahoo home page may be ranked poorly as it is rarely referenced. On the other hand, *Page Rank* (which is used in *Google* search engine) [47] propagates the weighting by recursively defining the weight of each page based on the weight of pages that point to it. The algorithm assigns a high weight to a page if it is pointed to by many other pages, or if some pages that point to it have high weights.

Web Coverage. Beside the ranking technique, a search engine's coverage of the Web indirectly reflects its retrieval performance. This coverage is commonly measured in terms of the quantity of indexable web pages. Lawrence and Giles [38] examined six search engines and found that no engine covered more than one-third of the Web (which was then estimated to contain 320 million pages). They proposed that web coverage could be improved by combining results of multiple engines.

Meta search engines send user's query to multiple search engines. The results returned are then processed and re-ranked (merged) before being presented to the user. Unlike conventional engines, meta search engines do not index web pages. Some examples of such engines are *MetaCrawler* [56], *SavvySearch* [28] and *ProFusion* [22].

The quality of the index is an alternative measure to the web coverage of search

engines. With the motivation that a small but high quality index might satisfy user needs on broad queries better than a large one, index quality was measured using an arbitrary page-weight assignment function [25]. A simple methodology that uses this function was devised to compare various engines by performing a random walk on the Web. It was found that indexes of search engines differ greatly in quality.

Performance evaluation. *Recall* and *precision* [45] are two measurements that benchmark the performance of retrieval techniques. The former measures its efficiency of retrieving relevant documents while the latter measures the relevance of the retrieved set of documents to the users' query.

$$\text{recall} = \frac{\text{Number pages retrieved that are relevant}}{\text{Total number of relevant pages}}$$

$$\text{precision} = \frac{\text{Number pages retrieved that are relevant}}{\text{Total number of pages retrieved}}$$

As data collections of search engines contain large number of web pages, the *recall* values are often hard to determine. However, *precision* can be estimated by examining the first N retrieved pages [24]. For example, precision at 20 of a search engine is:

$$\text{P@20} = \frac{1}{N = 20}(\text{number of the first 20 retrieved documents that are relevant})$$

In order to obtain a good measurement of the retrieval techniques, these values are averaged over a set of queries.

2.3.2 Query Expansion

When a search engine receives a query, it looks into its database of web pages for documents that contain the query terms. However, different words can be used to describe the same concept but authors often use the same word to represent the same concept in their document. This inconsistency causes a problem in information retrieval when searches are based on the co-occurrence of query terms in documents [63]. A common solution is to expand queries by including words or phrases of similar meaning to the query terms. *Relevance feedback* [2] iteratively reformulates the original query using the terms from relevant documents selected by the user. Given an initial query, the search engine retrieves a small number of documents, from which the user is to select the relevant ones. A new query, prepared from this set of selected documents, is then used to retrieve more documents. This process is repeated until a satisfactory set of relevant document is obtained.

Several automatic query expansion methods have been tested. An approach is to use a global thesaurus to add terms of similar meaning to the query [62]. *WordNet*[3] is an example of such thesaurus. It is an online reference system that maintains a database of words that are clustered based on their meanings. Instead of using a global thesaurus, a similarity thesaurus [49] that is based on the terms in the corpus can be automatically constructed. Such methods, that expand queries by examining the corpus as a whole, are known as global analysis techniques [63]. Local techniques, unlike the global ones, expand queries by examining the top ranked documents retrieved by the original query. Two such techniques were compared

[3] http://www.cogsci.princeton.edu/~wn/main/

by Xu and Croft [63]: local feedback and local context analysis. Local feedback is a method that automates relevance feedback by re-formulating the initial query using the terms obtained from the top ranked documents. These documents are returned by an arbitrary retrieval method using the original query. It was found that poorly-performed queries retrieve few relevant documents after local feedback since more words are obtained from non-relevant documents. Local context analysis is a technique combines global analysis and local feedback.

2.4 Link analysis in a hypertext environment

This section studies the framework of the Web. It aims to obtain a better understanding of the structure of the Web, in order to perform crawling and information searching on the Web more effectively.

Web communities. A web community is a set of pages that are related to a common topic. For example, web pages about cycling, swimming or rock-climbing might be found in a SPORT community. There are numerous, explicitly gathered, resource collections (e.g. the Yahoo web directories) in the Web that categorize URLs into a set of pre-defined topics based on their content. These topics are commonly ordered hierarchically. By observing link relationships between web pages and using a graph-theoretical approach, *implicitly-defined communities* can also be identified and enumerated automatically [37]. This is done by scanning web pages retrieved in a crawl, for web graph structures that are indicative signatures of communities.

Web pages may seem to be randomly inter-connected, however they do not form a worst-case graph as pages with related content are often linked together. Empirical evidence of topical locality in the Web and value of descriptive text as representatives of the targeted page was provided by Davidson [13]. It was found that the likelihood of linked pages having similar textual content is high.

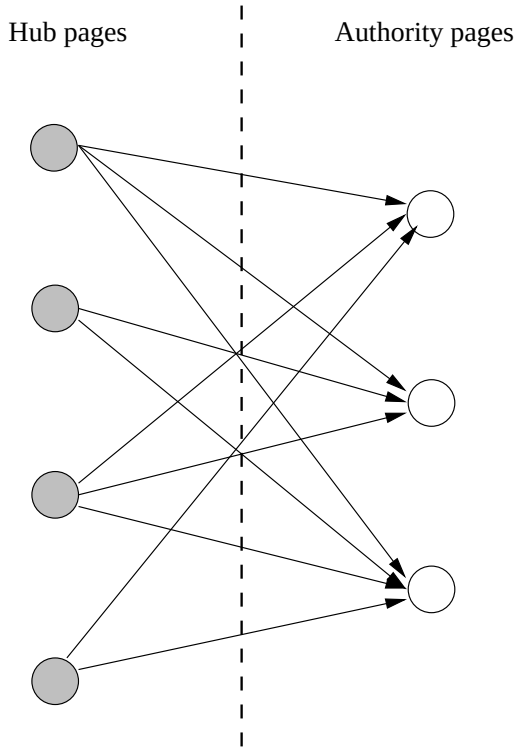

Figure 2.5: A conceptual example of a link relationship between hubs and authorities.

Hubs and authorities. An *authority* page is one that contains high quality information about a specific topic and a *hub* page is one that contains a large number of links to pages containing information about the topic [4]. The home

page of University of Waterloo is an example of an authority page for the topic "Educational Institutions". An example of a hub page is a car buyers' resource web site that contains links to many car dealer web sites. Kleinberg [34] observed that hubs and authorities have a mutually reinforcing relationship: a good hub often points to many authorities and a good authority is often pointed to by many hubs (see figure 2.5). Using this observation, he formulates an iterative algorithm that identifies authorities in a hyperlinked environment in the following manner [23]:

1. Each page p is associated with a hub $h(p)$ weight and an authority $a(p)$ weight, which are both initialized to 1.

2. The $a(p)$ of each web page is then updated by adding all the hub weights of every web page that points to p and $h(p)$ by adding all the authority weights of every web page that it points to. In other words, by denoting $p \rightarrow q$ as page p containing a link to page q, the weights of each page are updated by:

$$a(p) = \sum_{q \rightarrow p} h(q)$$

$$h(p) = \sum_{p \rightarrow q} a(q)$$

3. This process of updating is repeated until the authority and hub weights of all pages converge [23].

An implementation of Kleinberg's algorithm is HITS [23]. Given a user-query, HITS retrieves up to 200 pages returned by a search engine. This set of pages are

then increased by adding pages that points to the initial set and that the initial set points to. The algorithm is then applied on the set. The pages with the highest weights are then presented to the user. ARC [4] is a system developed at IBM Almaden Research Center that automatically compiles a list of authoritative web resources such as those provided by Yahoo. Its algorithm is an extension of HITS with slight-modification: ARC assigns authority weight of a page p by examining the text around the *href* links that point to p.

2.5 Web crawling

Web crawler is an agent that acquire resources, such as web pages and images, from the Web. Given an initial URL, the crawler (also known as a *web spider*) first retrieves the corresponding web page. URLs are then extracted from the page and stored into a work queue. This queue, which is not necessarily a FIFO, contains the URLs that are to be retrieved by the crawler. After a page has been processed, the next URL is removed from the work queue to repeat the retrieval process. There are many application of web crawling, for example indexing of web pages by search engine companies.

2.5.1 Web crawler anatomy

There are several issues that a crawler should address:

- **Robot Exclusion Protocol compliance.** Many web servers want to restrict crawlers from accessing certain, if not all, pages in their domains for

several reasons. For example, some pages (such as cgi-scripts) are not suitable for crawling. Thus the Robot Exclusion Protocol[4] was established. In order to specify the pages that incoming crawlers should not access, an access policy is specified in the "/robots.txt" file that is located at the top directory of a server. This protocol is not a standard of the internet, and it relies on total co-operation from the crawlers for compliance, hence it contains no security value to restrict web access.

- **Avoid web server overloading.** It is common for web pages to point to other pages within the same host. Thus there is a tendency for a crawler to request multiple pages from a host in a short period of time. This overloading of web servers should be avoided as it affects the servers' performance.

- **System recovery.** Crawlers often run over a long periods of time. Therefore they may break down for various reasons such as a shut down of the host for maintenance. When that happens, restarting crawlers with their initial URL seeds is redundant as retrieved data will be recollected. A self-recovering crawler is able to continue crawling from where it left off without having to re-crawl old pages.

- **Refreshing stale pages.** Information changes rapidly on the web, especially pages from the media (e.g. CNN pages) and financial domains (e.g. stock values). To keep a fresh copy of the web, these pages need to be re-visited more frequently than the others.

[4]http://www.robotstxt.org/wc/exclusion.html

- **Data duplication.** *URL* and *content duplication* are two issues that need to be addressed to reduce web retrieval redundancy. The former refers to the presence of a URL on multiple web pages. For example, many web pages link to the home page of Yahoo (http://www.yahoo.com), which is a common portal on the Web. The latter refers to different URLs that points to exactly the same page content. An example would be the following two URLs:

  1. http://plg.uwaterloo.ca/~c3chung/
  2. http://plg.uwaterloo.ca/~c3chung/index.html

  Duplicate web pages do not necessarily reside on the same host. For instance, Sun's version of the Java Documentation is likely to be available on numerous servers. Web crawlers that keep track of previously-visited URLs and previously-retrieved content are able to reduce repetitious work. However, unlike URL duplication, content duplication cannot be completely avoided since a web page needs to be retrieved before it is known if the page is a duplicate one. Once a page is identified as a duplicate, it is removed and its links will not be extracted for retrieval.

- **Prioritized crawling.** Exhaustive crawlers attempt to retrieve the entire Web and ignore the qualities of different web pages. However, not all pages are of the same quality. Some pages are more important because they contain information that the crawler is searching for, or perhaps they are referenced by more pages (i.e. high back-links). URLs of pages that are most likely to be important are retrieved first so as to keep a high quality harvest rate

throughout the crawl. Thus, the order in which web pages are visited is significant [9].

- **Bandwidth and server resources.** Considering the size of the web, crawlers require considerable system resources. If not supervised, they have the potential to overload the network's bandwidth or use up server resources such as disk space and CPU time.

2.5.2 Focused crawling

A rising interest in the area of web crawling is to train specialized crawlers to retrieve pages of particular interests [7, 16]. Brute-force crawlers rely on conventional graph algorithms, such as breadth-first or depth-first search, to traverse the Web. However, a crawler does not need to look into every corner of the Web if it is to acquire information on a specific interest. Therefore the crawler uses a topic classifier to follow links that are expected to point to pages on relevant topics. This is known as *focused crawling*.

Two methods of determining crawl routes have been proposed [7]: *soft-focus* and *hard-focus* methods. Both methods crawl a child page c only if some ancestor page of c is classified as a page of the relevant topic. The only difference is that the *hard-focus* approach selects child pages with the highest probability. A group from NEC presented a focused crawler that uses a context graph [16]. The classifier is trained by using the links and content of documents that are closely linked to a set of seed pages. It was claimed that the context-graph crawler outperformed the standard focused one by retrieving 50% to 60% more "on-topic" documents.

2.5.3 Distributed crawling

Distributed crawling addresses the issue of bandwidth and server resources in web crawling. It utilizes multiple web crawlers that are scattered across the Internet to retrieve web pages in a co-operative fashion [12, 60]. Experimental results shown a significant increase in retrieval rate when distributed crawlers are used [64]. Although the physical location of each crawler is immaterial to the functional correctness of a distributed web crawling system, they are situated away from each other so as to reduce bottleneck at physical nodes (e.g. gateways) of the Internet.

A collaborative crawling experiment was carried out at IBM Almaden Research Center [60]. They partitioned the Web based on a URL hierarchy and assigned crawlers to retrieve pages from each sub-space. If a link is retrieved that does not belong within the same sub-space, it will be routed to other crawlers. A major issue in collaborative crawling is to minimize such crossing links as the overhead for such transfer can be costly. Unfortunately no result was provided for their experiment. Their partitioning approach is based only on the URLs, and is independent from the content of the pages. Therefore URL duplication is a concern but they did not explain what they would do in this case.

Mobile crawling [19] is an alternative approach to a traditional web crawling system. It migrates the data retrieval component (i.e. the web crawler) to the host where the web pages reside. The mobile crawlers are managed by a crawler manager that resides in the home host of the crawling system. After the pages have been collected and processed remotely, they are transmitted back to the home host. This proposal reduces network bandwidth in several ways:

- Accessing web pages locally reduces request/response time.
- Unwanted pages do not need to be downloaded since the evaluation is done at remote hosts.
- Unwanted portion of web pages can be discarded before transmitting.
- Web resources can be compressed before transmitting over the Internet.

Although mobile crawling brings numerous advantages, several issues need to be addressed before the prototype can be used on the Web [36, 19].

Chapter 3

Distributed web crawling

Due to the size of the Web, multiple spiders are commonly deployed in a distributed fashion to retrieve web information [1]. One approach is to divide the Web into small partitions and assign a spider to crawl within each one. As pages often point to others in different partitions, retrieval redundancy may occur. Therefore web partitioning is a major consideration and is studied in this chapter from the perspective of web data duplication. The architecture of *Multitext* crawler, which was developed at the University of Waterloo, will be outlined next. Finally we present *C4*, a collaborative web crawling system that is adapted from the Multitext crawler. *C4* partitions the Web based on the content of pages. For the rest of the thesis, an entire distributed crawling network is referred to as a *web crawling system*, and each remote data retrieval component is simply referred to as a *web crawler* (or a spider).

3.1 Non-centralized distributed crawling

As the web grows, efforts to obtain a meaningful copy of it are becoming more tedious. In order to lighten the workload, multiple crawlers are often instantiated to perform web data gathering in a co-operative fashion [60]. However, the problems of URL and content duplication are amplified in distributed crawling systems. URL duplication occurs when the page of a URL is retrieved by multiple crawlers and content duplication occurs when the same web pages, referred to by different URLs, are retrieved by multiple crawlers.

URL duplication can be eliminated by using a centralized URL database server [12, 1]. The server maintains a list of URLs to be retrieved by the crawling system. Since this list is overseen by a single machine, it is easy to ensure that the same URL is not retrieved multiple times. To reduce content duplication, retrieved data needs to be maintained by a centralized server as well (see figure 3.1). However, the centralized approach creates a bottleneck and a single source of failure. To reduce these problems in centralized distributed systems (for example, file systems [27]) multiple servers are often connected in a hierarchical fashion (see figure 3.2).

An alternative to the above approach of distributed crawling is the non-centralized database. This approach divides (partitions) the Web among its crawlers; each crawler retrieves web pages from the partitions that are assigned to it. When a crawler retrieves a resource that does not belong to it, the resource is forwarded to its respective owner. There is no restriction on the number of partitions that can be assigned to a single crawler. However, to simplify the discussion, it is assumed that exactly one is assigned to each crawler. Hence the concepts of crawler and

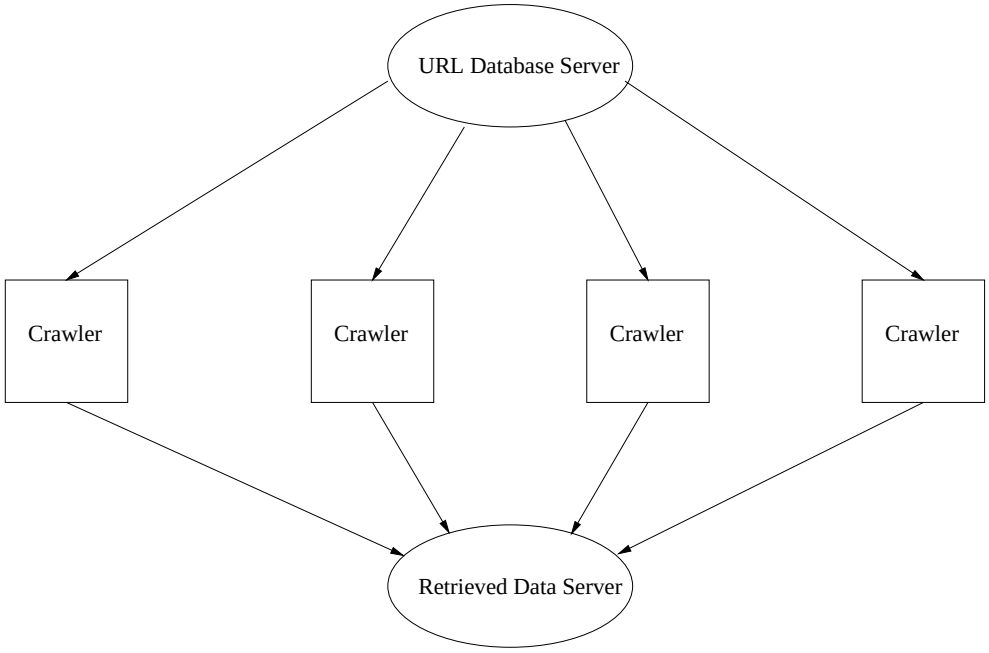

Figure 3.1: Centralized servers.

partition are closely associated in this chapter.

3.1.1 Web partitioning

There are pros and cons to this non-centralized approach. The concerns for bottlenecks and single source of failures have been replaced by issues of data duplication. Unlike the centralized approach, the approach chosen to partition the Web is more crucial here. Pages often point to others in different partitions, therefore a non-centralised crawler may retrieve pages that are outside its partition. If that happens frequently, then it is equivalent to having N crawlers retrieving a full copy of the Web. Various partitioning methods have been proposed and will be discussed next.

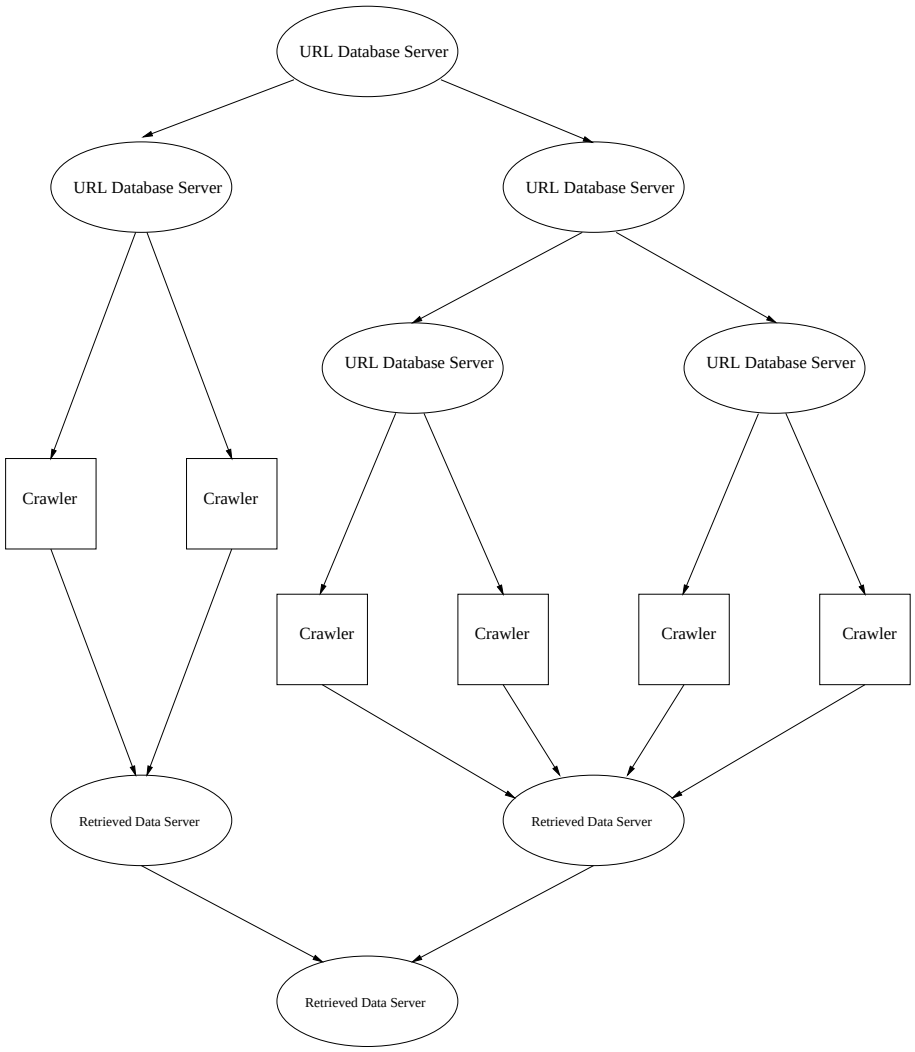

Figure 3.2: Hierarchical organisation of centralised servers.

URL-oriented partitioning URL-oriented partitioning methods are those that divide the Web by assigning URLs to partitions. They eliminate URL duplication as the same URLs are always retrieved by the same crawlers. The kind of resource that is transferred between the crawlers is the URL. To measure the effectiveness of various URL-oriented partitioning methods, we define the following measure:

Definition 3.1.1 *Degree of local URL assignment,* $\Pr_N^U(u)$*, is the probability of a crawler to retrieve the web page of a linked URL* u*, where* u *is extracted from a page retrieved by the crawler and* N *is the number of partitions.*

As the overhead for transferring URLs could be costly, $\Pr_N^U(u)$ should be maximized. Next, numerous URL-based methods are discussed.

A common URL assignment method is *URL-based hashing*. It hashes URLs to determine their partition, hence pages referred to by the same URL are retrieved by exactly one crawler. A good hash function produces an unbiased random partition assignment, therefore $\Pr_N^U(u) = \frac{1}{N}$.

Web pages may be more likely to refer to those that are found within the same host. For example, the web pages of Yahoo directories contain numerous links to other pages residing in the Yahoo site. *Host-based hashing* ensures that URLs from the same host are retrieved by the same crawler by hashing only the host name, instead of the entire URL [8].

Country domain takes this approach step further by examining only the country domain extension of each URL (e.g. "br" for Brazilian's servers and "sg" for Singapore's). Pages whose host names end with the same country domain extension are located in close proximity geographically. By assigning crawlers to retrieve pages that are located physically close by, the Internet's traffic-load and data transfer time can be reduced. For example, a crawler located in Japan is assigned to retrieve pages whose URL host names end with "jp" (which is the country domain extension for Japan). Although work bas been done on geographical crawlers [12, 64], they are implemented using the centralized database approach.

The *URL hierarchy* approach is a variant of URL-based and host-based hashing. It divides the Web by breaking down a URL into sub-domains. This dissection of URL does not need to end at the host name, it can be further fragmented down the directory. A group from IBM Almaden Research Center [60] implemented a URL hierarchy partitioning approach that dynamically alters the division of Web among the crawler, in order to perform load-balancing.

Content-oriented partitioning URL does not reflect the content of its associated page. Since a web page can be referred to by different URLs, URL-oriented partitioning would result in multiple crawlers obtaining copies of identical web pages. However this redundancy can be reduced by partitioning the Web based on the content. URL-oriented and content-oriented partitioning methods differ in the order in which the partition assignment is determined with respect to the retrieval of web pages. The former assigns a partition before the web page is retrieved (pre-retrieval assignment) and the latter assigns a partition after the retrieval (post-retrieval assignment). Due to this dissimilarity, the same measurement for partitioning effectiveness cannot be use on both kinds of methods. To measure the effectiveness of various content-oriented partitioning methods, we define the following

Definition 3.1.2 *Degree of local page assignment,* $\Pr_N^P(\delta)$*, is the probability of a crawler assigned to retrieve the links in the page* δ*, where* δ *resides in the same partition as the crawler and* N *is the number of partitions.*

An example of such approach is *content-based hashing.* When a crawler is given a URL, it retrieves and hashes the URL's web page content. The hash value in-

dicates the crawler that is to retrieve the child pages of the links from the page. To avoid repetitious connections to HTTP servers (which are expensive operations in web crawlers), the retrieved web content (instead of the URL) is transferred to the corresponding crawler. This hashing approach increases the amount of data transfer between the crawlers over the URL-based ones, however it reduces content-duplication. A good hash function produces an unbiased random partition assignment, therefore $\Pr_N^P(\delta) = \frac{1}{N}$.

Combination partitioning Both *content-based* and *URL-based hashing* complement each other's redundancy issues: the former eliminates URL duplication but fails to address content duplication while the latter ignores URL duplication but reduces content duplication.

An alternative hashing approach, *combination hashing*, is to hash on both the URL and its content. Each crawler is assigned ownership to a set of URLs and a set of web pages. These sets are assumed to be independent. In other words, a crawler owns a URL does not imply that it owns the corresponding web page, and vice versa. When a URL is received by a crawler A, it is checked if it has been seen. The URL will be removed if so, or the corresponding web page δ will be retrieved. The page δ is then hashed to determine its owner (say, crawler B). After B obtains δ from A, it will discard δ if the page has been previously seen. If not, each link from δ is extracted and hashed to determine the crawler that has ownership over the URL. The retrieval process is then repeated after these links are distributed to their crawlers. This approach has not been implemented but has the potential to reduce the redundancies faced in *URL-based* and *content-based hashing*. However,

the superiority of this approach comes with a high price tag as large amount of data needs to be transferred.

3.2 Multitext crawler

The *Multitext crawler* is a multi-threaded, single processor web spider developed at the University of Waterloo. The crawler has been used to generate crawls of up to a tera-byte of web data at a rate of up to 600 KB/sec.

3.2.1 Design of the crawler

The crawler contains multiple independent child processes that operate on batches of URLs. These batches are processed in ordered stages. They are also processed concurrently; multiple processes may be working on multiple batches at each moment.

The child processes are co-ordinated by a *crawler manager*. The manager begins by reading a crawler script that specifies parameters, such as the format for input and output filenames, associated with each process. At each stage, a child process reads in a set of input files and produces a set of output files, which may then be treated as the input files for another process. The input files are removed after being processed by the program if the scripts specifies so. A process is instantiated by the manager if an input file for that program exist. However, no two instances of the same process may be ran at the same time. The crawler begins crawling with an initial URL batch. After the last processing stage, more URL batches are produced. These batches are fed back into the crawler to repeat the retrieval process. No

assumption is made by the manager about its child processes. This brings about a modular design to the crawler, enabling the crawler to be highly extensible; processes can be added into Multitext crawler to implement new functionality just by changing the script.

System recovery Instead of circumventing the various possible faults that can break a crawler, the Multitext crawler takes on a passive approach and focuses on failure recovery.

When a process is started, it is assigned a URL batch and a unique transaction number. This number will be used as the suffix on the output files generated by the process. A transaction log file is updated to indicate the start of this process with this transaction number. The successful termination of processes are also indicated in the log file. When a Multitext crawler begins execution, it first reads the log file (if it exists) to search for processes that failed to commit during the last crawl. Existence of such processes implies that the crawler was terminated abruptly. The batches that correspond to these processes are rolled back to their initial state by removing the files with the transaction number as its suffix and are restarted during the next crawl.

3.2.2 Tasks of child processes

The following list describes the independent child processes of the Multitext crawler (see figure 3.3):

- The *robot resolver* performs two tasks. It implements the Robot Exclusion

Protocol and maintains a DNS cache that maps host names to IP addresses (see figure 3.4). This cache reduces the time for resolving host names. Besides the DNS cache, there is a robot cache that stores information about robots.txt file of each host. Both caches are updated together. In other words, if an IP address of a host exists in the DNS cache, then its robots file would be found in the robot cache. When the resolver is given a URL, it searches for the host's IP address in the DNS cache. If it does not exist or is more than a day old (i.e. it has expired), the host name cache and the robot cache will be updated. After update of the robot cache, the URL is checked against the cache to determine if the crawler is restricted from accessing this web page. Restricted URLs are removed from the batch.

- The *fetcher* starts multiple threads to retrieve web resources of a URL batch. Given a URL, each thread connects to the HTTP server where the resource resides. After the connection is established, the thread sends a resource request to the server. The connection is closed by the server after the resource has been transferred. After the batch is fetched, the retrieved web resources are merged together.

- The *extractor* generates a new set of URLs by extracting links (URLs) from the retrieved web pages by looking into HREF or SRC attributes of every HTML tag. If a relative URL is specified, it is resolved to its absolute form. At this point, crawled data is output from the crawler.

- *Xdup* removes duplicate copies of web pages. The crawler keeps track of dupli-

cate pages by hashing the content of each page (excluding the HTTP header) using the *MD5* [50] message digest algorithm. Given a string of arbitrary length, MD5 produces a 128-bit signature. This algorithm is commonly used to verify data integrity. If a page produces a hash value that has been seen by the crawler, it is treated as a duplicate page and is discarded.

- The *restrictor* removes the URLs that are indicative of non-HTML pages (e.g. those that ends with GIF or JPEG). It also restricts the crawler from accessing pages that do not belong to a specific domain, For example, the crawler might only want to retrieve pages that are within the University of Waterloo domain.

- The *filter* maintains a database of URLs that have been retrieved and removes URLs that has been previously crawled. Thus, it addresses the issue of URL duplication.

- The *scrambler* shuffles a URL batch to reduce retrieval of web pages from the same host simultaneously, thus avoiding web server overloading.

- *Batch* is the last process to operate on each batch of URLs. It distributes the remaining URLs into several smaller batches that are to be fed back into the crawler.

3.2.3 System considerations

The Multitext crawler offers options to limit bandwidth utilisation. The crawler attempts to maintain a user-defined rate of incoming data by dynamically controlling

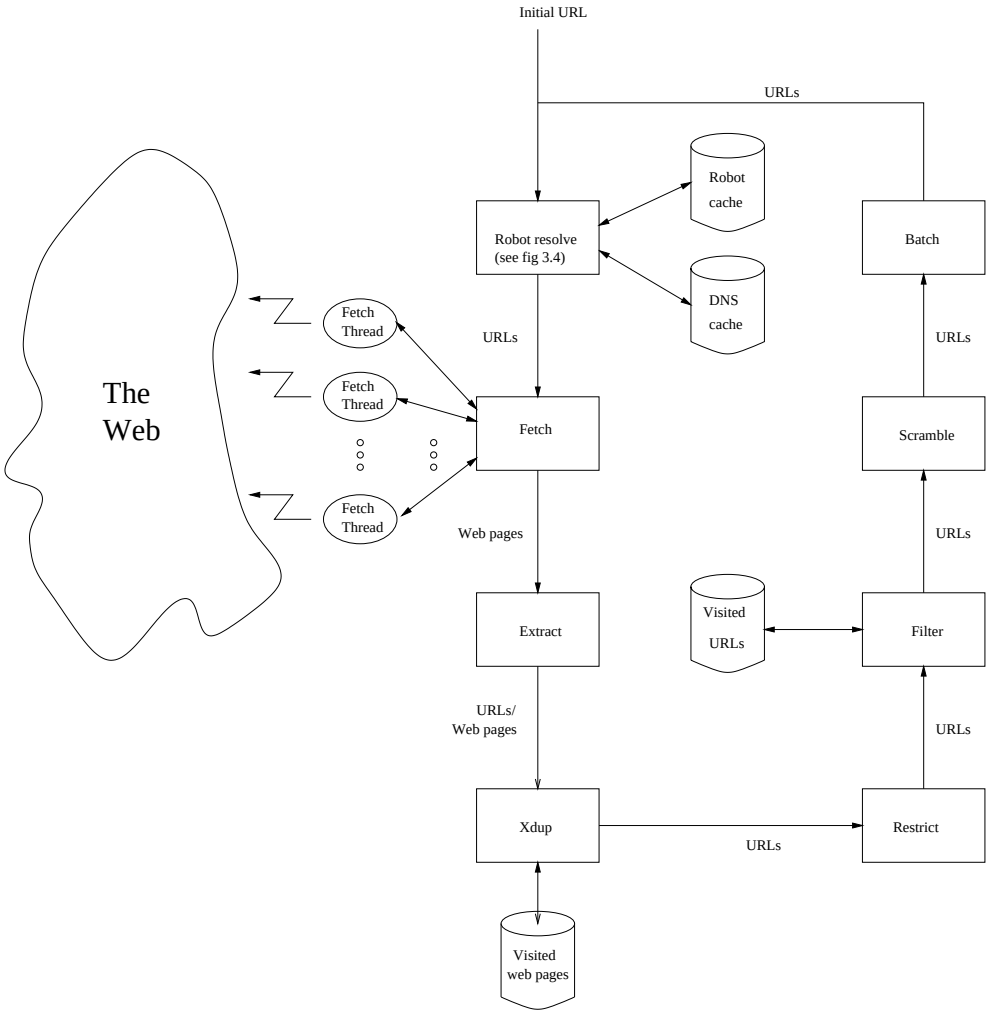

Figure 3.3: High-level view of the design of Multitext crawler.

the number of fetch threads. This prevents the crawler from dominating network bandwidth. Crawlers can also be set to idle for a period of time each day. When a crawler is idling, it has no active child processes. This is to prevent the crawler from retrieving data in the daytime when the network is shared by many users.

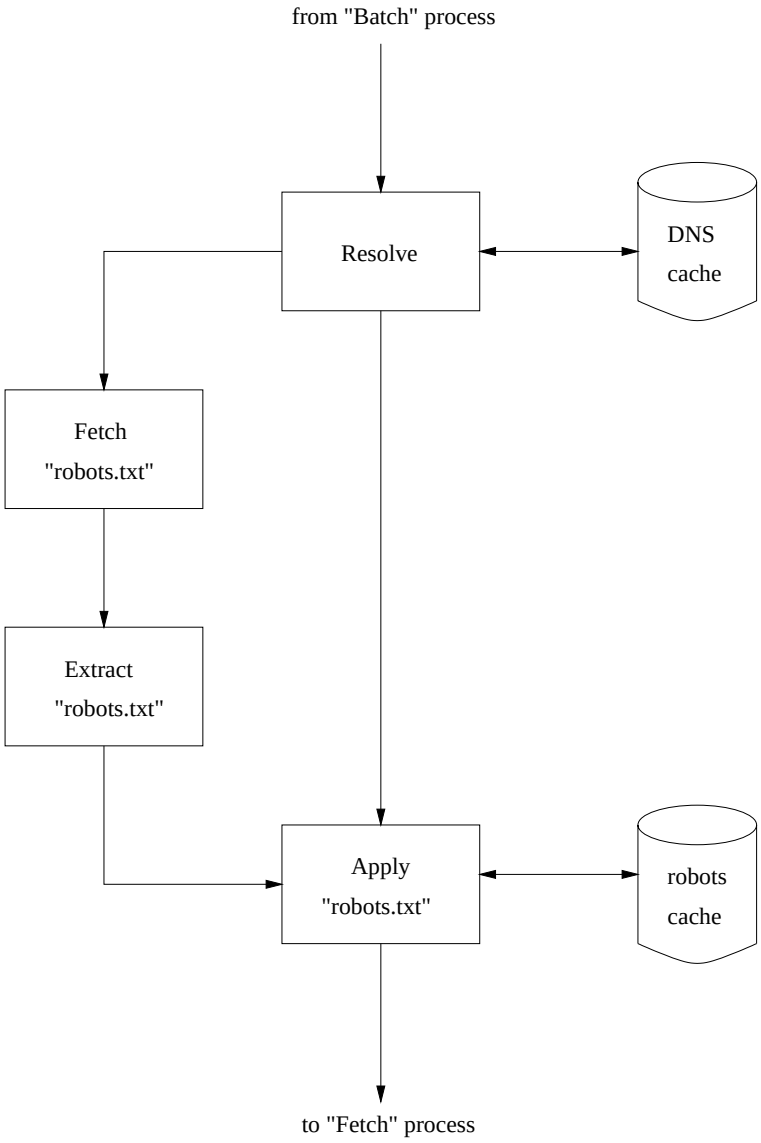

Figure 3.4: Resolve process diagram.

3.3 The topic-oriented crawling system

We present C4, a novel approach to web data gathering. It is a non-centralized distributed web crawling system, adapted from Multitext crawler, that divides the

Web using a *topic-oriented partitioning* approach.

3.3.1 Topic-oriented partitioning

When a spider of a distributed crawling system enters a page, it retains the resources (URLs or web pages) that belong to the spider and route the rest to their appropriate crawlers. Davidson [13] found that web pages are significantly more likely to link to pages that are topically related, as opposed to pages that are selected at random. Thus crawlers that crawl pages on a specific topic may be more likely to continue retrieving pages on that topic. These spiders are commonly known as *focused crawlers*. By dividing the Web into various categories of general interest, we propose *topic-oriented partitioning*, a novel approach to web-partitioning for distributed web crawling.

A spider is assigned to crawl web pages of a specific topic. When it retrieves a web page, it examines the HTML content to determine the topic that the page is most relevant to. If it is most relevant to the topic that the spider is assigned, it will be retained. Otherwise, it will be routed to the appropriate spider. For example, a spider that has the category of *ART* assigned to it focuses on retrieving pages that have artistic content. If it comes across a page that contains *BUSINESS* content, the page will be transferred to the *BUSINESS* crawler. We call this form of distributed crawling *collaborative*.

This partitioning approach allows the system to be highly scalable. Crawlers may be added by breaking topics into sub-topics. For example, the category of BUSINESS might be divided into technological and non-technological industries.

Crawler boundaries When a crawler in *C4* retrieves a page that does not belong to it, we say that the crawler has reached a *boundary*. One of our main concerns is to determine the frequency of a crawler reaching its boundaries. A crawler that encounters its boundary frequently is inefficient as it needs to route a large amount of web pages to external crawlers.

3.3.2 Architecture of C4

C4 is our implementation of a web crawling system that uses the *topic-oriented partitioning* approach. This system uses Multitext crawlers to retrieve web pages from each topic. It allows multiple instances of Multitext crawlers to execute independently and provides a corridor to allow interaction between them. To achieve this behaviour, two processes are added into each Multitext crawler: *classifier* and *gofer* (see figure 3.5).

From a set of categories, the classifier determines the most relevant one to assign to an arbitrary web page. This set of categories is determined by the categorized corpus used to train the classifier. The task of *gofer* is to transfer pages between the spiders in the crawling system. Next we discuss these two processes in details.

C4 classifier

Given a web page (including the HTTP entity header), the classifier examines its HTML text to determine its category. This category indicates the crawler that has ownership over this page. If the page belongs to a remote crawler, then it will be stored in a local directory waiting to be picked up by the gofer associated with the

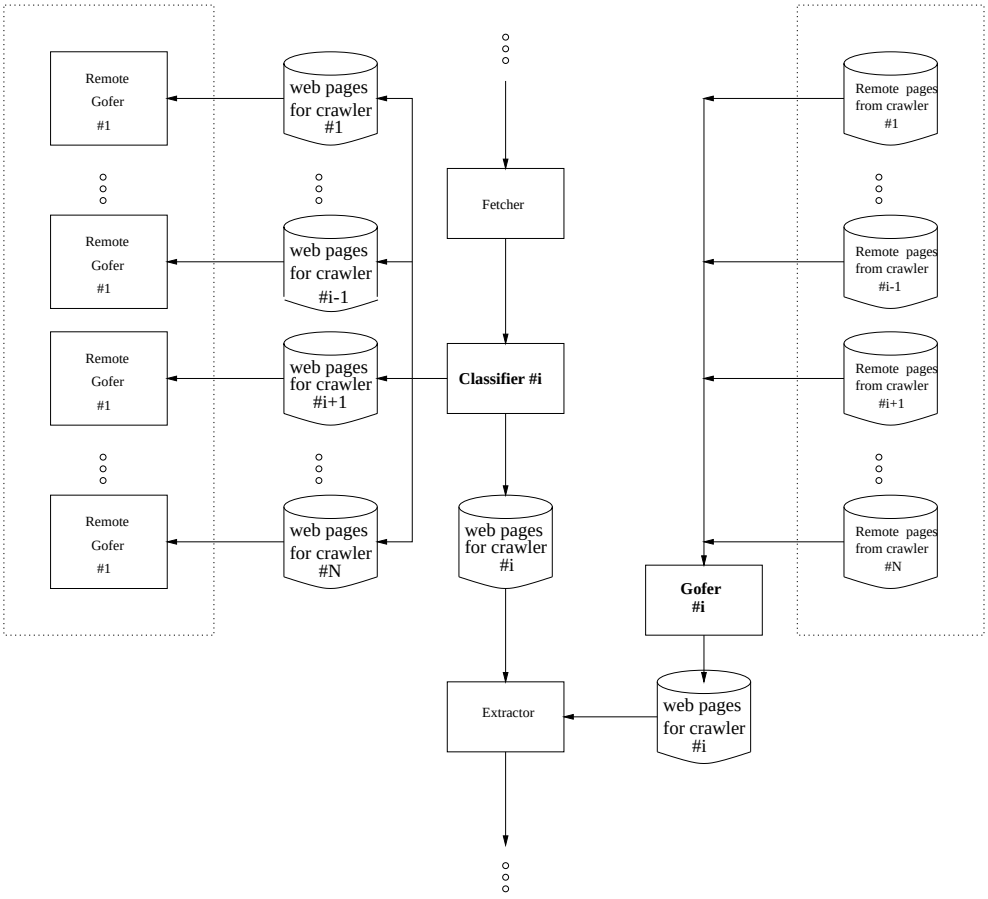

Figure 3.5: C4 classifier and gofer processes for crawler #i.

crawler. Otherwise, it will be retained locally so that its links will be retrieved and classified by the local crawler.

Although it is possible, not every web page should be assigned to a category based on its content, as some pages contain HTML text that are not indicative of any topic. We define a page to be unclassifiable if it is

- **EMPTY**: a page that has no HTTP header and no HTML data.
- **INVALID**: a page that does not have a 2XX HTTP status code.

- **NO-TEXT**: a valid web page that does not contain any informative HTML text. Non-informative text includes punctuation, numerical data and white space.

These pages, which will not be classified by the classifier, are identified in the pre-processing stage (see figure 3.6). During this stage, categories are assigned to them but not based on its content. Empty pages are assigned to the local crawler (category) because, otherwise unnecessary overhead is required for transferring these URLs without any benefit gained (since they have no out-going links). Invalid web pages, which are mostly re-direct (3XX) and client error (4XX) pages, are also be retained locally because:

- Re-direct pages often contain URLs where the resources actually reside. Therefore unnecessary overhead is required if these pages were routed to an external crawler since they will often be immediately routed back.

- Other form of invalid pages seldom contain URLs. Routing these pages introduces unnecessary transfer overheads.

Valid web pages that do not contain any HTML text are commonly found in the Web. To avoid duplicate copies of web pages from residing in different crawlers, these unclassifiable web pages are not retained locally. Instead they are arbitrarily assigned to category #0.

If a web page is classifiable, it is fed into the topic classifier for categorization. The classifier plays an important role in *C4* as a crawler with an ineffective classifier reaches its boundary frequently. The objective of the next chapter is to identify a

classifier that is able to divide the Web effectively so that spiders in our collaborative crawling system have minimal interaction.

Gofer

The crawlers interact on a pull protocol. In every crawler, there is a directory for storing batches of out-going pages for each remote (external) crawler. After pages have been retrieved and classified, those that are assigned to remote crawlers are stored in their appropriate directories (see figure 3.5). To perform the transfer, every crawler has a *gofer* process that looks into its corresponding remote directories for these batches of web pages. A transfer sequence, which is performed every 30 minutes, involves the copying of pages from the remote directories, follow by the removal of these pages if they are successfully transferred. Transferred web pages are then merged. As these are categorized pages, they will not be reclassified, instead they are directly fed into the extractor (see figure 3.5).

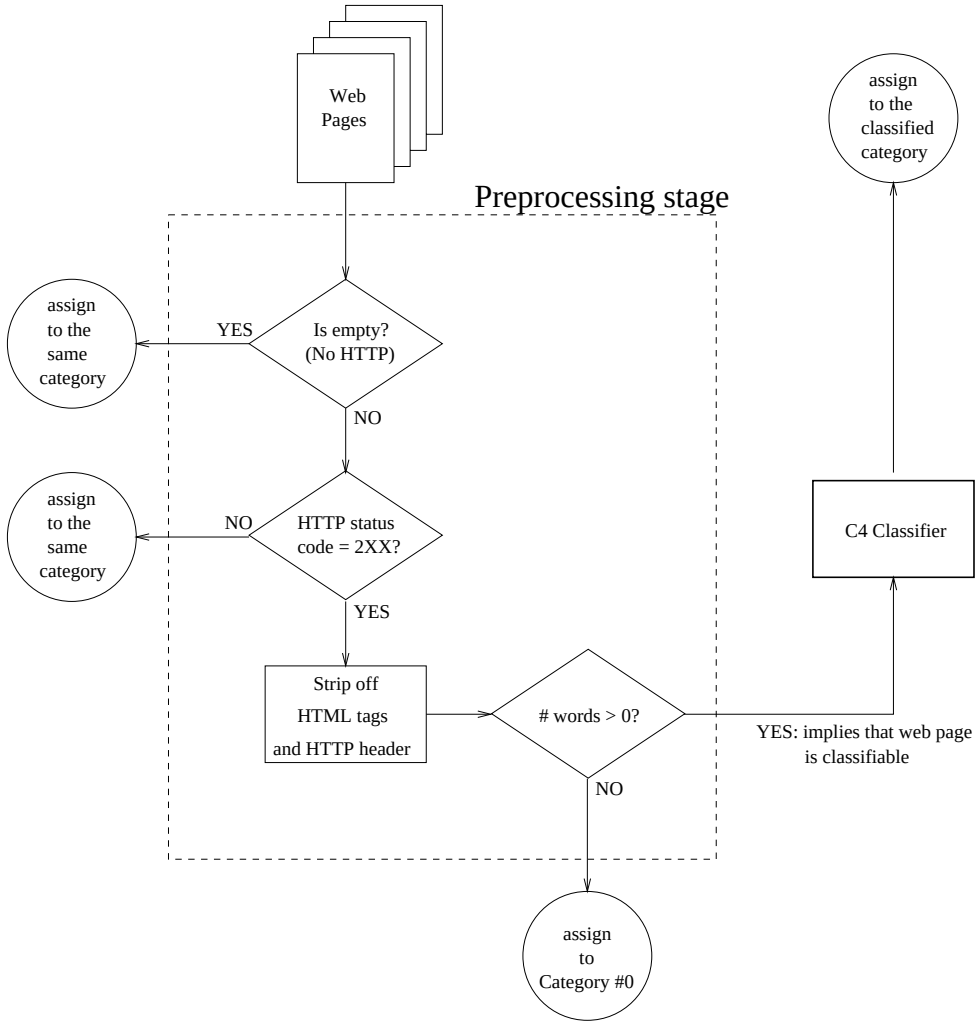

Figure 3.6: Pre-processing stage of C4 Classifier, in which unclassifiable web pages are identified and are assigned with categories.

Chapter 4

Classification of web pages

The objective of this chapter is to identify a classifier for our collaborative crawling system. The Web is divided into a set of topics, with a crawler assigned to retrieve pages from each one. When a page is retrieved by a crawler, it is examined by the classifier to determine its topic. This topic indicates the crawler that is to retrieve the links from the page. As the classifier is to be used on web data obtained in real time, its components have to be effective and efficient.

Text categorization is the grouping of text documents into a set of predefined categories [30, 41]. Since a main focus of our work is to classify web pages into a set of topics, an area of text categorization that is of importance to our crawling system is topic classification. Topic classification is the problem of assigning a topic to a document based on its content. For example, given only EDUCATION and SPORT categories, a university course web site may be more likely to belong to the former.

The Web contains documents in many natural lanagues. The natural language

of an arbitrary web page needs to be determined before topic classification can be applied. Therefore, another aspect of text categorization that we focus on is language identification, which is the problem of identifying the language in which a document is encoded.

The focus of this work is on English language pages. Instead of discarding non-English pages, these pages are grouped into a single category. At the end of this chapter, a topic classifier and a language identifier is combined to form the *C4 Classifier*.

4.1 Classification background

Text Categorization is the problem of assigning predefined categories to text documents based on the probabilities suggested by a set of training documents. The words in a document are extracted and weighed against the training data using a classification method. The category that gives the highest probability is the one assigned to the document. Given a large set of new documents, a classification system replaces human judgment by automatically assigning categories to them, thus producing a large database of categorized documents to support effective and efficient data retrieval.

The evaluation of a classifier involves two stages: training and testing. Training a classifier requires a large set of pre-classified documents, in which good discriminators of each category are identified. A good discriminator of a category is a term that characterizes the category. For example, the words "gallery" and "university" are good discriminators of ART and EDUCATION categories respectively. After the

classifer has been trained, it is tested by re-assigning categories to another large set of pre-classified documents. The performance of a classifier is measured by the number of documents that are correctly classified in the testing stage.

4.1.1 Corpus

The corpus to be used as training and testing data needs to be selected before a classifier can be evaluated. It consists of a large set of documents that are classified into different categories. Publicly available test collections are gaining popularity as a common source of corpora to be used for research in text retrieval, because they eliminate the time needed for data preparation (e.g. converting unclassified data into classified data) and promote comparison between different implementation of classification techniques. Some of the publicly available corpora are:

- The **Reuter Test Collection** is a corpus of full-text newswire stories from 1987 that are classified into more than 100 categories. The most updated collection contains 21,578 news articles (therefore known as Reuter-21578) and is publicly available[1]

- The **OHSUMED**[26] corpus was developed at Oregon Health Sciences University to promote medical information retrieval research. It has 18,000 categories containing over 300,000 medical documents.

- The Web harbours many **online hierarchical collections**. Each online collection contains URLs that are categorized into a set of hierarchical topics.

[1] http://www.research.att.com/~lewis/

These collections are commonly used to train and test classification techniques on Web pages [17, 5].

An example of such a collection is the **Open Directory Project** (ODP). ODP is a self-regulated organization maintained by volunteer experts who categorize URLs into hierarchical class directories. At the top level, there are 21 categories. Volunteers examine the content of each URL to determine the class (category) that it belongs to. Each level in the hierarchy contains a list of URLs of relevant topics and a list of categories for the next level. This categorized platform can be accessed though the Web[2]. Alternatively, the entire directories of categorized URLs can be down-loaded[3].

4.1.2 Term representation

A corpus normally consists of a large collection of documents. Various representation issues of these documents need to be resolved before they can be used as training and testing data for a classifier. Each document is broken up into individual words known as *terms*. These terms are not necessarily individual words and may also be phrases in the document, such as "New York".

Term weighting

A weight is allocated to each term to reflect its importance to a document. Let $W(t,\delta)$ be the weight of a term t in a document δ. Numerous weight assignment

[2]http://dmoz.org/

[3]http://dmoz.org/rdf.html

schemes for W(t,δ) exist and are presented next.

Term frequency A boolean indication of whether a term is in a document is the simplest form of term weighting scheme:

$$\mathrm{W}(t,\delta) = \begin{cases} 1 & \text{if } t \text{ is found in } \delta \\ 0 & \text{otherwise} \end{cases}$$

But this representation fails to differentiate the importance of each term that appear within a document. Hence the more frequent terms in a document are given the same weight as the uncommon ones. *Term frequency* (TF) is an improvement over boolean indication by observing that more significant terms are more frequently found in a document.

$$\mathrm{W}(t,\delta) = \mathrm{TF}(t,\delta) = \text{number of occurrence of } t \text{ in } \delta$$

Inverse Document Frequency (IDF) *IDF f*ocuses on the occurrence of words across the collection of documents such that rarely-appearing words in the collection are more important. Hence common words, such as "the" and "of", are ranked low in the list. Many variants of *idf* have been introduced [32, 10]. For instance, the *idf* of a term t in the collection can be given by

$$\mathrm{IDF}(t) = \log\left(\frac{\text{number of documents in the collection}}{\text{number of documents containing the term } t}\right)$$

A term weighting scheme that combines *tf* and *idf* was proposed by Salton [54]:

$$\mathrm{W}(t,\delta) = \mathrm{TF}(t,\delta) \times \mathrm{IDF}(t)$$

Weighted Inverse Document Frequency A disadvantage of *idf* is that it does not distinguish the frequency distribution of terms across the collection of documents. Consider a collection of D documents such that a term t' appears once in every document except in δ^*, in which t' occurs numerous times. The *idf* of t' (which computes to zero) is not very informative since the distribution of t' across the collection is very skewed. Motivated by this observation, *Weighted Inverse Document Frequency* (WIDF) [61] was proposed by using the frequency of each term in the documents:

$$\mathrm{W}(t,\delta) = \frac{\mathrm{TF}(t,\delta)}{\sum_{i \in D} \mathrm{TF}(t,\delta_i)}$$

4.1.3 Document indexing

Indexing documents in a large corpus can be a computational expensive procedure. The *term vector model* treats each word as an indexable term and the order in which the words occurred is ignored. Each document is represented by a high-dimensional vector of words (known as document vector) in a vector space. Let $T = \{t_1, t_2, \ldots, t_{|T|}\}$ be the set of terms (words) that appear in a corpus, where t_j is the indexable term j, then document i can be represented as $\delta_i = \{W_{i,1}, W_{i,2}, \ldots, W_{i,|T|}\}$ where $W_{i,j}$ is the weight of term j in document i. With this model, documents can be indexed more efficiently[58]. The *term frequency vector model* is a common form of document representation that uses *tf* to denote the weight of each terms in the

documents.

Phrase indexing addresses the co-occurrence of words in a document by observing the phrases. Each phrase, which is a term made up of multiple words, is considered to be in a document if all the words in the phrase occur in it. Fagan [18] compared syntactic and non-syntactic phrasal indexing representation and found that the the latter performs better.

Both *word* and *phrasal indexing* methods assume that a query is related to a document based on common words. Retrieval is based on matching the terms in the queries to those in the indexed documents. These techniques are also known as *surface-based matching* [66]. However these word-based techniques fail to address two fundamental issues of word characteristics: *synonymy* and *polysemy*. *Synonymy* is the fact that multiple words can have (nearly) the same meanings (e.g. "car" and "automobile") and *polysemy* refers to words that have multiple distinctive meanings (e.g. "Java" can either be a programming language or an island of Indonesia). *Latent Semantic Indexing* (LSI) [15] addresses these deficiencies by using singular-value decomposition to arrange documents in a document space. It observes the latent semantic of words to position documents with similar conceptual content in close proximity. After the documents have been indexed, the retrieval is carried out by using the query terms to identify a single document, which is then returned to the user along with its neighbouring documents.

4.1.4 Classification technique

Only after various term and document representation formats has been determined, can classification be performed. Numerous methods have been proposed for classifying text documents and are reviewed in this section. In particular, three classification methods that we have compared experimentally will be studied in detail. These methods are chosen because they are easy to implement. We believe that if these simple methods prove to be effective for the collaborative crawling system, then more complex algorithms will be worth exploring.

Naive Bayes Classifier

Let C be a space with K mutually exclusive and exhaustive partitions such that $C_1 \cup C_2 \cup \ldots \cup C_K = C$ and for every $i \in \{1, \ldots, k\}$, the prior probability of category C_i is greater than zero (i.e. $\Pr(C_i) > 0$). *Bayes' Theorem* defines the probability of assigning an instance δ to partition $C' \in C$ as:

$$\Pr(C'|\delta) = \frac{\Pr(C')\Pr(d|C')}{\sum_{i=1}^{k} \Pr(C_i)\Pr(\delta|C_i)}$$

To find the most probable category C^* assignment to δ, $\Pr(C^*|\delta)$ needs to be computed for every category. Hence the classification decision rule can be simplified to

$$C^* = \max_{i=1}^{k} \Pr(C_i)\Pr(\delta|C_i)$$

Naive Bayes text classifier (NB) is commonly studied in text categorization [44]. The naive aspect of the classifier, which is the assumption that terms found in a

category are independent from each other, greatly improves the efficiency of the classification. The decision rule of NB states that

$$C^*(\delta) = \max_{i=1}^{k} \log \left\{ \Pr(C_i) \prod_{w \in \delta} W(w, C_i)^{n(\delta,w)} \right\}$$

where C^* is the category assigned to a new document δ, $W(w, C_i)$ is the weight of a term w in category C_i and $n(\delta, w)$ is the number of times w occurs in the document. A variation of NB that has been studied uses a binary model on $n(\delta, w)$ [35].

Lewis' approach

Lewis [40] extended a probabilistic text retrieval model (proposed by Fuhr [21]) to text categorization. He proposed that $\Pr(C_j = 1|\delta)$, the probability of assigning category C_j to document δ, can be estimated by

$$\Pr(C_j = 1) \times \prod_i \left(\frac{\Pr(W_i = 1|C_j = 1) \times \Pr(W_i = 1|\delta)}{\Pr(W_i = 1)} + \frac{\Pr(W_i = 0|C_j = 1) \times \Pr(W_i = 0|\delta)}{\Pr(W_i = 0)} \right)$$

where

- $\Pr(C_j = 1)$ is the prior probability of category C_j.
- $\Pr(W_i = 1)$ is the prior probability that feature W_i is present in a randomly selected document.
- $\Pr(W_i = 1|C_j = 1)$ is the probability that feature W_i is assigned to a document given the knowledge that C_j is assigned to that document. $\Pr(W_i = 0|C_j = 1) = 1 - \Pr(W_i = 1|C_j = 1)$

- $\Pr(W_i = 1|\delta)$ is the probability that feature W_i is assigned to document δ. $Pr(W_i = 0|\delta) = 1 - Pr(W_i = 1|\delta)$

Rocchio-TFIDF

Relevance feedback is a common approach to information retrieval by improving a query through user feedback (see section 2.3.2). Rocchio Relevance Feedback [51] uses term frequency (TF) and inverse document frequency (IDF) to determine the weight of each query term. Numerous variations of the Rocchio alorithm have been proposed. The variant that is tested here is defined as follows [30]: let $TF(w, d)$ be the term frequency of the word w in the document δ, F be the set of features and the inverse document frequency $IDF(w) = \log \frac{|D|}{DF(w)}$, where $|D|$ is the size of the document and $DF(w)$ is the number of documents in the training data that contains w. The decision rule of assigning the category C^* to a new document δ' is

$$C^*(\delta') = \max_{c \in C} \frac{\sum_{w \in F}(TF(w, \delta')IDF(w))(TF(w, C)IDF(w))}{\sqrt{\sum_{w' \in F}(TF(w', C)IDF(w'))^2}}$$

Other classifiers

Support Vector Machine. *Support vector machine* (SVM) was initially targetted at pattern recognition, but has been gaining popularity in the area of classification [17, 31]. In short, it defines a hyperplane in a document vector space to separate the positive documents from the negative ones. Platt [48] proposes an alternative algorithm for training SVM: Sequential minimal optimization (SMO). SMO improves the efficiency of the classification by breaking a large quadratic

problem in the SVM into a series of small quadratic problems that can be solved analytically.

K Nearest Neighbour (kNN). The algorithm of kNN [65] is straightforward. Given a document δ, it finds the k most similar documents from the training data. The category of δ is then determined from the categories assigned to these documents. The weight of each category is obtained by summing the similarity scores between these neighbouring documents and δ. The category with the highest weight is then assigned to δ.

Linear Least Squares Fit. LLSF [66] is an example-based mapping method for document retrieval and classification. Given an initial set of queries, documents are selected using a *surface-based matching* retrieval technique (4.1.3). Words from each set of retrieved documents are added to the associated query, which is in turn used to obtain more related documents that were not retrieved by the inital query. A LLSF technique therefore may be used to determine the related documents of arbitrary queries.

4.1.5 Feature engineering

After a classifier has been trained, each category is characterized by a set of terms. These terms are known as *features*. Due to noises in the training data, these features may contain bad discriminators that affects the performance of classifiers. Therefore the feature set is often put through a series of selection and extraction steps to amplify good category discriminators, while removing bad ones. Various feature

engineering processes have been investigated, including straightforward approaches such as the removal of stop words (e.g. "the", "of", "are" etc) and the least and most frequent words.

- **SFS** and **SBS** [29] modify the feature set by examining the features one at a time. Stepwise Backward Selection (SBS) temporarily removes a feature and then evaluates the classifier with the reduced set. If the deleted feature does not deteriorate the classifier performance, it will be permanently removed. This process is then repeated for every feature. Stepwise Forward Selection (SFS) works in the reverse order. A feature is added into an increasing feature set if it improves the classification.

- **Combined Stepwise Selection** (CSS) [55] incorporates *SBS*, *SFS* and *exhaustive pairwise search*. The pairwise search method is similar to both SBS and SFS methods except that the classifier is tested on all subsets of two features. It takes into consideration the fact that many features occur in pairs (e.g. "New York").

- **Document Frequency** [67] considers the number of documents in which each feature is found. It assumes that features that appear in more documents are more important.

- **Term strength** [68] estimates the importance of each feature by observing its likelihood of appearing in sets of closely-related documents.

4.1.6 Document clustering

Text categorization is the classification of new documents into a set of pre-defined categories. Therefore, categorized corpora are needed to train classifiers to identify the categories of documents. Such corpora are uncommon but can be built from uncategorized ones by manual inspection of each document. However, this approach is time-consuming, thus it is undesirable. An alternative to manual inspection is to cluster documents automatically based on their similarity values. Each document is given a value based on its content. Unlike text categorization, clustered documents are not associated with predefined topics, clustering merely groups documents. Various methods have been adopted to determine the similarity values of documents:

- **Word/Phrase-intersection** [70] groups documents that contains words (or phrases) that are shared by all documents in the cluster.
- **Locality-Sensitive Hashing** [59] hashes similar values to similar documents.
- **Term clustering** [33] is the grouping of terms with similar meaning.
- **Term clustering of syntactic phrases** [42] is a clustering method proposed by Lewis that combined both term clustering and syntactic phrasal indexing.

Conventional clustering methods are based on the words found in the documents. However link analysis has been applied to document clustering on the Web using two algorithms [14]. These algorithms take the URL of a page to retrieve a

set of related Web pages based on the connectivity information and not the content of the pages. The *companion* algorithm identifies authority pages that are in close locality to a source page. It first builds a vicinity Web graph of the input page, then uses a modified HITS algorithm to retrieve the top authority pages in the graph. The *co-citation* algorithm is an alternative approach to the above. It returns the pages that has the highest degree of *co-cited* nodes. Two pages are *co-cited* if they are referred to by a common parent page. Pages of similar interest tend to have high *degree of co-citation* (number of common parent pages).

4.2 C4 Topic Classifier

The objective of this experiment is to determine a classification technique that not only classifies web pages accurately and efficiently, but also tends to classify a child page (link) in the same category as its parent. Considering the diversity of web content, the above mentioned points are not neccessary the same.

Three classification methods are compared: Naive Bayes (NB), Lewis and Rocchio-TFIDF. These methods are chosen because they are simple and have low computational complexities (O(n), where n is the number of words). Prior probabilities required by the first two classification methods are assumed to be uniform across all categories.

4.2.1 Training and testing data

The Open Directory Project (ODP) corpus is selected for training and testing our classifier. It is chosen because its categorized data (web pages) are very similar

to our target platform (the Web). A snapshot of 673 MB of data was obtained from the ODP in November 2000. It contains URLs that are classified into their corresponding categories. For the purpose of our experiments, the entire URL directory tree is collapsed into the top categories, 17 of which contains categorized URLs (i.e. are non empty). Among these non-empty categories, the content of two of them were found to be overly unfocused for our classifier:

- the REGIONAL category contains web pages that are specific to various geographic areas. It organizes sites according to their geographic foci and relevance to particular regional populations. For example, web pages about the climate survey in the arctic and those about Arab Human Rights in the Middle East can be found in this category.

- the WORLD category contains sites of non-English languages. Instead of using the classification technique to identify non-English pages, the language identifer uses a different approach that does not require training data. However, this category is used to test the identifier. The language identifier is explained in detail in section 4.4.

Hence only fifteen categories are chosen (see table 4.1).

Adult	Arts	Business
Computers	Games	Health
Home	Kids/Teens	News
Recreation	Reference	Science
Shopping	Society	Sports

Table 4.1: Top categories in ODP chosen to be used in the English topic classifier.

Data selection criterion. After the categories have been determined, we proceed in the following manner to obtain the data for testing and training the classifier. After removing non-http URLs in all the categories, the remaining URLs of each category are randomly divided into two equal and disjoint sets. 500 web pages are then retrieved from these sets. Each page has/is

- HTTP tag of "content-type: text/html".
- a valid HTML page (i.e. 2XX connection status code. See section 2.1).
- more than 50 words, excluding HTML tags and scripts.
- identified as English page. The identification technique used is discussed in section 4.4.

These retrieved pages totalled to 227 and 232 MB respectively (see table 4.2 for the breakdown).

Term extraction. Before a web page can be processed, it must be broken down into tokens. There are three kinds of tokens: HTML tags, scripts and text. As classification of the web pages is to be based only on the text, our HTML document parser ignores the HTTP header, and removes all HTML tags and scripts. Numerical data are removed as well. The remaining text is tokenized into words based on white spaces and punctuations. These words are then converted to lower case. The parser also attempts to interpret common HTML entity characters that can be represented by 7-bit ascii codes (e.g. "<" is interpreted as "<" and "&" as "&"). The other entity characters are treated as whitespace.

Category	Training Data Size (MB)	Testing Data size (MB)
Adult	11.3	12.8
Arts	13.5	13.6
Business	10.7	11.1
Computers	14.8	16.7
Games	13.5	12.9
Health	17.6	17.1
Home	25.2	26.4
Kids/Teens	12.8	13.8
News	12.2	12.2
Recreation	12.4	12.1
Reference	21.1	20.1
Science	18.3	18.5
Shopping	12.8	12.4
Society	15.1	15.6
Sports	15.9	17.0

Table 4.2: Size of training and testing data gathered for each category.

Category characterization. Each category is characterised by a list of terms extracted from the training data web pages. Each term is associated with two weight values: the number of term occurrences in the category and the number of documents the term appears in.

4.3 Classification experiments

Naive Bayes (NB), LEWIS and Rocchio-TFIDF are the topic classification techniques that were compared. As the final classifier is to be used in our collaborative crawling system, its requirements are different from the conventional ones which focuses only on accuracy. The classifiers were compared by conducting three different

experiments:

1. *Classification Methods Comparison* inquired about the classifiers' performance on different categories.

2. *Hyperlink Relationship* was an experiment we designed specifically for this thesis. It looked into the performance of classifiers in the Web hyperlink environment. We want to identify a classifier that would categorize a page and many of its links under the same category.

3. *Classification throughput* identified the classifier that is able to categorize the most amount of pages in the shortest time.

4.3.1 Classification Methods Comparison

To carry out the first experiment, 500 testing pages from each category were fed into the three methods for classification. The result is shown in figure 4.1. The means and variances of precision of the methods are computed:

	NB	LEWIS	ROC-TFIDF
Average	60.7 %	58.2 %	43.7 %
Variance	212.6964	356.8354	125.9911

The low mean value of ROC-TFIDF excludes the technique out of the competition. Between NB and LEWIS techniques, the former achieved 2% higher accuracy than the latter, with a smaller variance.

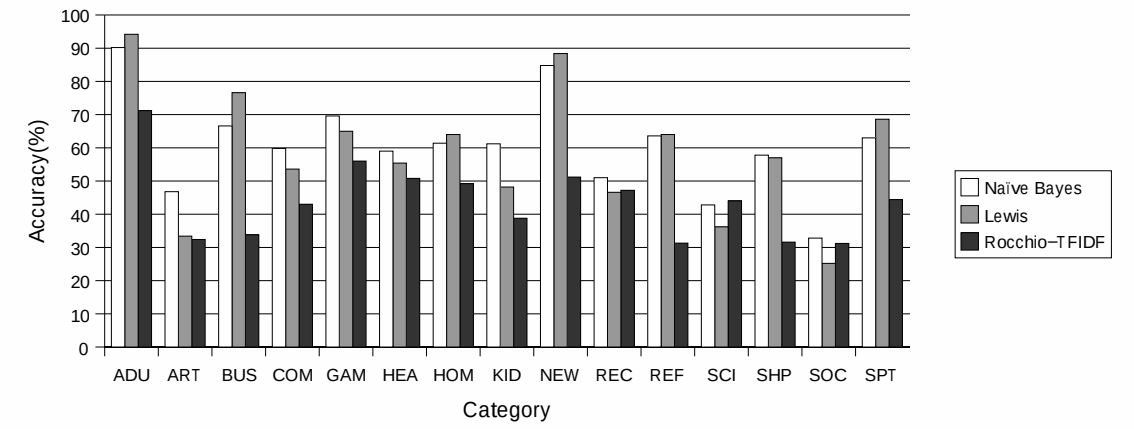

Figure 4.1: Performance of classifiers on different categories.

4.3.2 Hyperlink Relationship

This experiment investigated the performance of each classifier in a hypertext environment, particularly the probability of routing a child page to another crawler. As a crawler gathers pages from the web, it needs to determine which category each page should be assigned to. That, in turn, determines the crawler assigned to retrieve and classify the links from the page. Classifiers that assign many child pages to external categories generate heavier overhead to perform the transfer of these pages, thus they are considered inefficient.

100 web pages were randomly selected from each category in the testing data. Up to five random links from each page were retrieved and classified. We recorded the number of pages that were classified under the same category as their parent pages. This likelihood is shown in figure 4.2. The average and variance of this likelihood was also computed:

	NB	LEWIS	ROC-TFIDF
Average	62.4 %	62.3 %	48.9 %
Variance	101.6592	70.7912	85.1478

NB and LEWIS performed competitively in this experiment. An average link classified using one of these methods had a probability of 62% of assigning the same category as its parent page. ROC-TFIDF performed relatively poorly at only 49%.

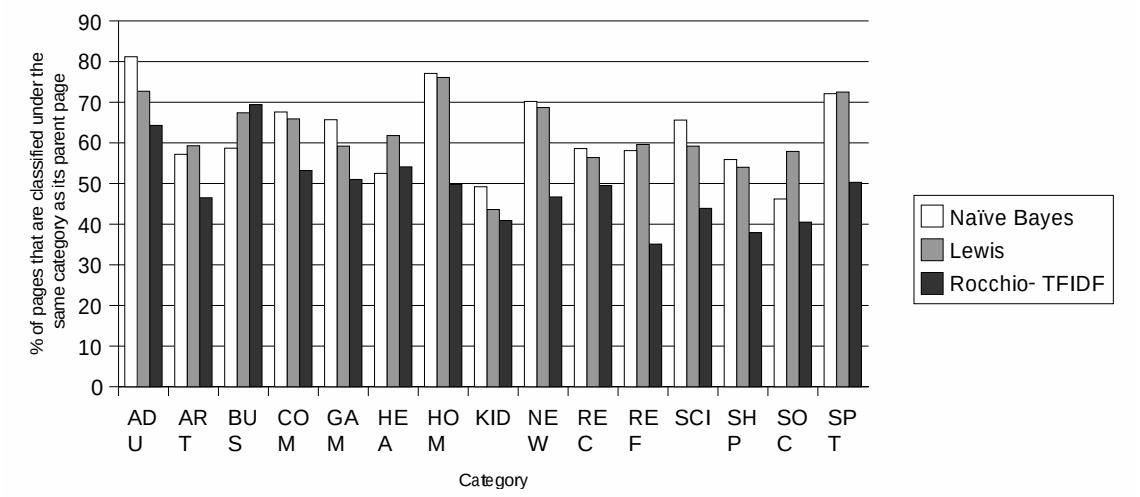

Figure 4.2: Likelihood of a link to be classified under the same category as its parent page.

4.3.3 Classification throughput

The classification will be performed on web data obtained in real time by the distributed crawling system. An inefficient classifier is a potential bottleneck that slows down the crawling process. This experiment investigated the throughput of each classifier. 30,000 web pages with an average length of 7,738 bytes each were

fed into the classifiers. The following table showed the number of web pages the classifiers categorized in a second:

Category	NB	LEWIS	ROC-TFIDF
Throughput (pages/second)	19.50	2.89	1.61

LEWIS performed almost twice as fast as ROC-TFIDF's, but NB significantly outperformed the rest; NB classified almost 12 times more pages than ROC-TFIDF's each second.

4.3.4 Discussion

NB produced the best result in all three experiments and ROC-TFIDF produced the worst. Although NB and LEWIS performed competitively in the first two experiments, it is the third experiment that mades NB the most attractive classifier as it was able to categorize web pages substantially faster than the other two. Hence NB was selected for use in our collaborative web crawler.

4.4 Language Identification

Given an arbitrary web page, the language in which the page is presented needs to be identified before data processing can be performed on it. This can be determined by looking at the HTTP character set field ("char-set") and the HTTP language fields, however they are often unavailable. Even if they are available, the language specified may not be consistent with the web content.

Multilinguality is a major concern in this research. Web pages that are of non-English content are referred to as foreign pages. As our topic classifier is trained

on an English corpus, using it to categorize foreign pages would produce erroneous results. Therefore, our crawlers need to determine if a page is of English content before it can be classified into topics. There has been exhaustive research in the area of language identification; current identifiers have a correctness of over 90%. Despite this encouraging result from these identifiers, we propose a fast and simple algorithm since our requirements for a identifier subtlely differ from the classical ones:

1. Classical identifiers attempt to recognise multiple languages while ours identifies only English pages. Therefore implementing an identifier for multiple languages is not only overkill, but it is also computational costly.

2. Classifical identifiers train and test on a pre-defined set of languages. The results of these identifiers when categorizing languages that they are not trained on were rarely (if ever) made available. However the Web contains many languages using various encoding scheme (character-sets). Using the classical approach to identify a random page obtained from the Web, the classifier has to train on web pages of all encoding schemes. Such training data ia hard to obtain. Therefore, the target corpus for our identifier contains languages not found in the training data.

3. Most identification techniques are concerned only with the accuracy and ignore the efficiency of the algorithms. However, the language identifier, like the topic classifier, has to perform on an enormous amount of web data obtained in real-time. Hence it not only has to be accurate, but also efficient.

Despite these differences, comparing the results of multiple language identifiers to that of ours provides a good reference as to how well our identifier performs.

4.4.1 Classical language identification techiques.

Several approaches to language identification are introduced next.

Unique character string. A simple approach is to look for substrings in words that are characteristical to a language [57]. For example, the string *czy* is unique to Polish. The task is to identify unique string sequences among all the languages. However, unique substrings are hard to find as the language domain increases.

Common words approach. Another simple approach to language identification is to observe the common words of various languages [11]. For example, the frequent occurrence of words such as "the" and "of" highly suggests that it is an English text. Likewise, *el* and *de* would suggest Spanish. However the performance of this approach deteriorates as the text size reduces.

N-gram frequency profile. *N-gram* is an approach that takes a step further with "unique character string" by slicing a string into a set N-character contiguous sub-strings. For example, the various n-grams of word "GAME" (with padded blanks ("*") at the head and tail of the word) would be:

bi-grams: {*G, GA, AM, ME, E* }

tri-grams: {**G, *GA, GAM, AME, ME*, E** }

quad-grams: {***G, **GA, *GAM, GAME, AME*, ME**, E*** }

Cavnar and Trenkle [3] constructed a n-gram frequency profile from the training data. It was tested on 3,713 language samples, which averaged at 1,700 bytes. The results produced an accuracy of over 90%.

4.4.2 Language identification on the Web

Classical identifiers are able to identify only languages that they are trained on. Hence the results of recognising unknown languages are not mentioned in many papers. This poses a problem on the Web where languages are diversified.

A language identification web spider was implemented by a group from New Mexico State University [11]. Like the classical identifiers, this one identifies a (large) set of predefined languages. The algorithm used by the spider was a variable n-gram approach, which was an extension from the method proposed above by Cavnar and Trenkle. Their error rate ranges from 11.92% (for 20-byte pages) to 0.27% (for 1000-KB pages).

4.4.3 Design and performance

A list of English words is available in a file ("/usr/dict/words") on every Unix operating system. In Linux Redhat version 2.0.36 (which was the operating system used for our experiments), the file contains 45,402 distinct words. The words were sorted in increasing order, ignoring upper and lower case. Certain Unix platforms (such as SunOS version 5.6) exluded words that end with the letter "s" if the same word without the last letter "s" was already in the list. For example, "success" and "make" were found in the list, but "makes" was not.

Using this list of words, we designed a simple method to determine if a web page is of English content. A brute-force approach was taken to parse web pages by assuming that the characters adhered to US-ASCII encoding scheme, however characters that were encoded in invalid forms (non-zero most significant bit) were not interpreted. In other words, characters with an ascii value of more than 127 were processed as they were.

Let w be a word in a document δ. Define $w^{\overline{s}}$ as the word w with the last "s" character (if it exist) removed, and let E(w)

$$E(w) = \begin{cases} 1 & \text{if } w \text{ or } w^{\overline{s}} \text{ was found in the word list} \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

$$\Pr(\delta \text{ is English}) = \frac{1}{|\delta|} \sum_{w \in \delta} E(w) \tag{4.2}$$

Proportionality Cutoff K. First we needed to determine the *Proportionality Cutoff* K for our identifier such that the document δ is an English page if $\Pr(\delta) > K$. If K was too large, we would be rejecting many English pages. Conversely, we would end up classifying many foreign pages if a small K was used.

Sixteen categories were chosen from ODP. Of these categories, only one (WORLD) is of foreign content, while the rest (see table 4.1) are of English content. The identifier is tested on 500 web pages selected randomly from each category. Figure 4.3 shows the number of pages classified as English. The following table shows the precision of the identifier at three values of cutoff K:

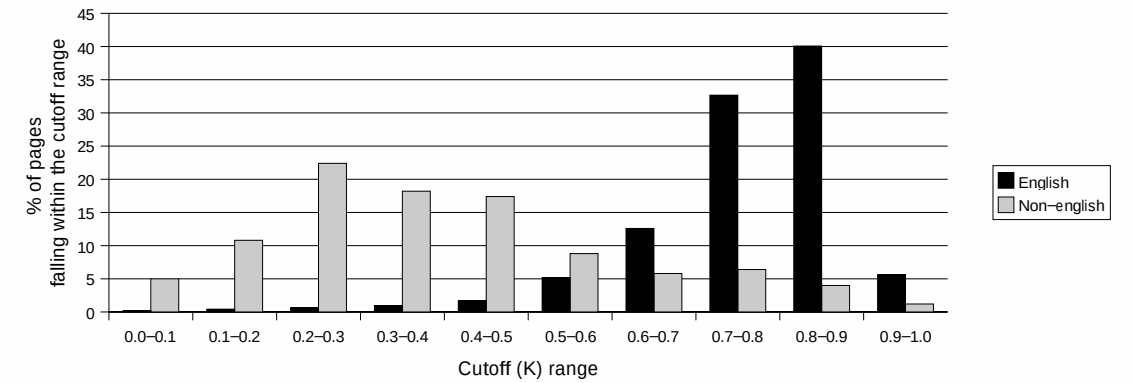

Figure 4.3: Distribution of $\Pr(\delta)$ across different ranges of cutoff K

	$K = 0.4$	$K = 0.5$	$K = 0.6$
English categories	98.1 %	96.1 %	90.9 %
Foreign category	56.4 %	73.8 %	82.6 %

The optimal value of K should produce high precision on both the English and the Foreign categories. K at 0.4 is a bad cutoff as over 40% of the foreign pages were classified wrongly. By increasing K from 0.4 to 0.5, the loss of 2% over English pages is expiated with a gained of 17% over the foreign ones. However from 0.5 to 0.6, it loses 5.2% for a gain of less than 10%. Hence K is set at 0.5. After removing noise from our training data (such as English pages that are in the non-English category), the identifier has an accuracy of 90.22% and 96.1% on identifying foreign and English pages respectively.

Despite its simplicity, the identifier is able to produce high accurarcy.

4.5 Components of the C4 Classifier

A language identifier and a topic classifier that are efficient and effective have been presented in this chapter. The identifier and classifier are combined to form the C4 Classifier (see figure 4.4). When the classifier is given a classifiable web page, it uses the language identifier to determine if the web page is of English content. If it is, it will proceed to classify the web page into a pre-defined set of topics using the topic classifier.

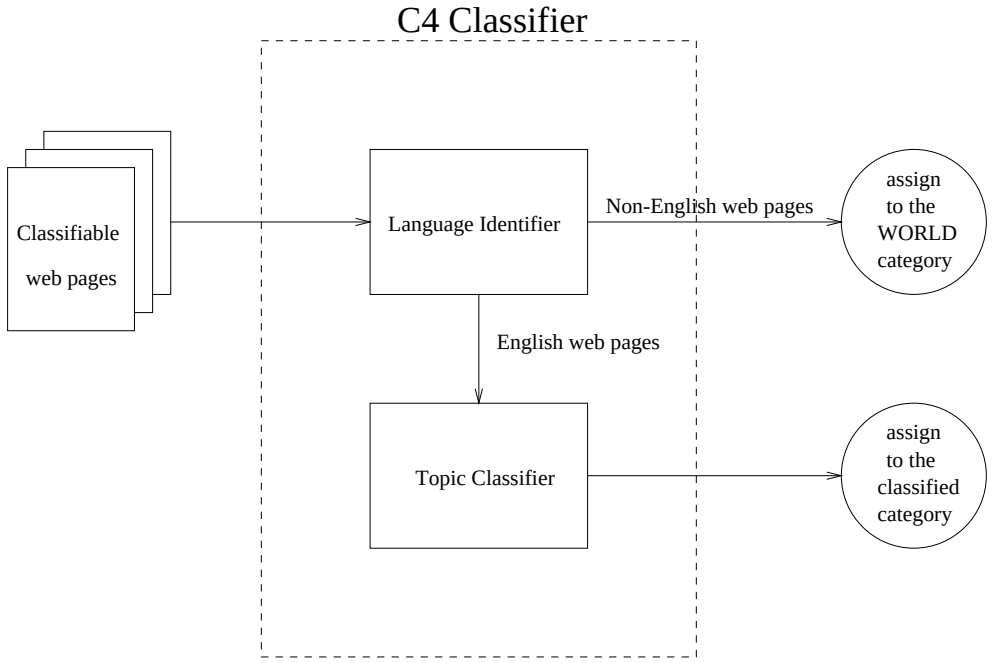

Figure 4.4: Categorization of classifiable web pages by C4 Classifier.

The C4 Classifier is trained to recognise a total of sixteen categories, of which fifteen are based on the topics of interests (see table 4.1). The sixteenth category contains pages of non-English content and is named as the WORLD category (note

that the training data for this category is not associated with the WORLD category in ODP).

Chapter 5

Experimental results

This chapter presents the data gathering results of *C4*. The collaborative web crawling system retrieved web pages for about a month. The experimental setup is described in the first section, followed by the results. Finally, a discussion on these results is presented.

5.1 Experimental setup

The collaborative crawling system deployed sixteen instances of the Multitext crawler. Each crawler was assigned to retrieve web pages of a particular category and no two crawlers were assigned to the same category. When a web page was retrieved by a crawler in the system, the crawler verified that it was a classifiable web page. If it was, the crawler proceeded to categorize it into one of sixteen categories using the *C4 Classifier*. Of these categories, fifteen contained English pages of various topics (see table 4.1). The sixteenth (WORLD category) contained foreign pages.

Hardware specification. The distributed setup for *C4* was simulated over a cluster of six workstation servers. Each server had a 300MHz processor with 128 MB of RAM, and ran Linux kernel version 2.0.36. Of these six servers, five executed three crawlers and one a single crawler. The following table shows the categories that were installed on each server:

Server	Categories assigned
A	ADULT, ARTS, BUSINESS
B	COMPUTERS, GAMES, HEALTH
C	HOME, KIDS, NEWS
D	RECREATION, REFERENCE, SCIENCE
E	SHOPPING, SOCIETY, SPORTS
F	WORLD

As the servers were connected to the University of Waterloo shared network, each crawler was set to retrieve data at a rate of 16 KB/sec to prevent it from dominating the university's external bandwidth. Retrieval was also limited to six hours at night when the network traffic is low. When the execution time ended each morning, the active processes were not terminated abruptly. Instead they were allowed to run to completion but no new processes were spawned. A crawler is said to be idling if it has no child processes running. Therefore, during that period, it retrieves no web data.

5.2 Results

5.2.1 General retrieval results

The crawling experiment was carried out over a period of 28 days, spanning across May and June 2001. Throughout this period of time, *C4* was not retrieving data constantly. Besides idling in the daytime, the crawlers were shut down periodically for maintenance. Whenever a crawler is resurrected from a shut down, it would continue its retrieval from where it was interrupted. In the month of June 2001 when *C4* was running, the university sent 1052 GB and received 2148 GB of data over its external links. Of these, 0.005 GB and 123 GB (5.7%) were sent and received by the crawlers.

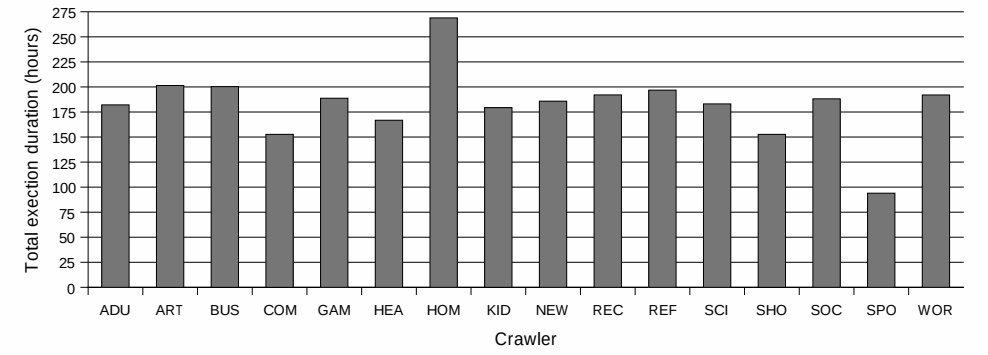

Figure 5.1: Duration of the crawlers in execution mode.

Child processes were spawned and terminated constantly in each crawler. A crawler is considered to be active or in its *execution mode* when it has at least one living child process. Figure 5.1 shows the amount of time the crawlers were active during the experimental period.

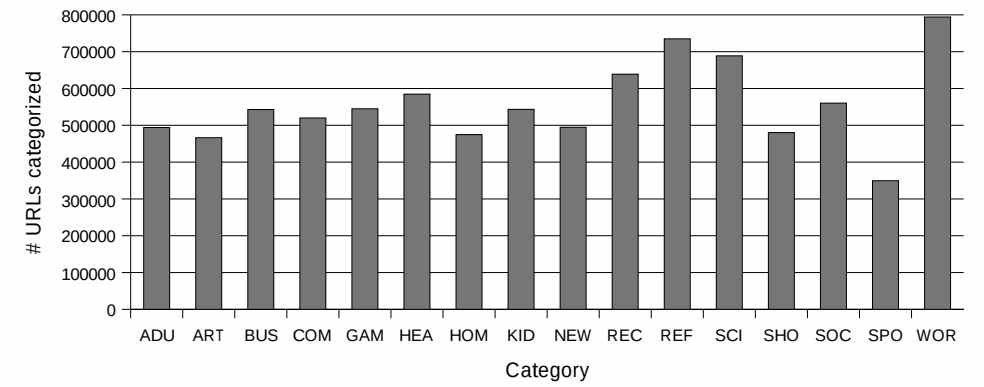

Figure 5.2: Number of URLs retrieved by each crawler.

URLs retrieved. A total of 8.912 million URLs were retrieved by the *C4 Classifiers*. These pages totalled to 137.36 GB of data for an average of 16.12 KB/page. Figure 5.2 and 5.3 shows the distribution of these data across the crawlers.

5.2.2 Topic classification

After the pages were retrieved, they were fed into the pre-processing stage of the topic classifier before they were categorized based on their HTML text. In this stage, unclassifiable pages were identified.

Unclassifiable pages. The unclassifiable web pages were identified during the pre-processing stage. These pages fell into three groups: empty pages (i.e. no HTTP header), invalid pages and valid pages that had no text. A total of 2.111 million web pages, 23.69 % of all the URLs retrieved, were identified as unclassifiable. These pages summed to 5.57 GB of web data, 4.1 % of all the web data retrieved. The distribution of these unclassifiable pages over the categories is showed in table 5.1.

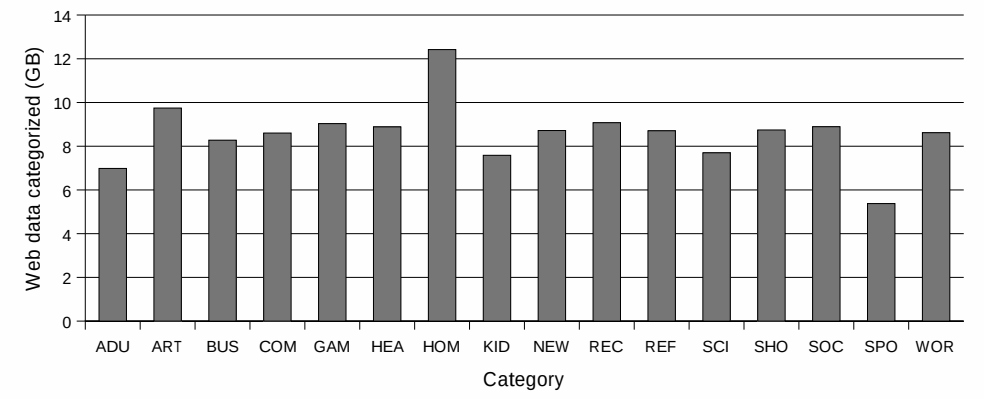

Figure 5.3: Amount of web data retrieved by each crawler.

43.0% of these unclassifiable URLs were empty pages, 14.9% contain no text and the remaining 42.1% were non-2XX (invalid).

Classifiable pages. Classifiable web pages constituted 131.79 GB of data (95.9 %). Categories were assigned to these pages by the topic classifier based on their content. The probability that a URL in a category will remain in the same category as its parent page is presented in figure 5.4. Appendix A and B shows the amount of URLs and web data distributed by each crawler respectively.

5.2.3 Duplicate retrieval by crawlers

Duplicate pages are those that are retrieved by more than one crawler. The URLs of these pages are found in web pages of various topics. Figure 5.5 shows the amount of duplicate pages retrieved by multiple crawlers.

Category	Empty	No Text	Invalid
ADU	47263 (0)	157304 (2.809)	42213 (0.167)
ART	47292 (0)	5896 (0.066)	56422 (0.056)
BUS	58290 (0)	12183 (0.140)	47067 (0.051)
COM	57660 (0)	7910 (0.122)	55643 (0.065)
GAM	52384 (0)	11617 (0.114)	64153 (0.070)
HEA	58088 (0)	5775 (0.052)	44518 (0.043)
HOM	21922 (0)	4218 (0.054)	36223 (0.056)
KID	55608 (0)	8296 (0.086)	57661 (0.056)
NEW	59689 (0)	7176 (0.106)	46598 (0.061)
REC	52140 (0)	9917 (0.095)	55635 (0.048)
REF	94957 (0)	10511 (0.135)	69602 (0.069)
SCI	79479 (0)	7415 (0.105)	60427 (0.047)
SHO	42565 (0)	8883 (0.084)	55140 (0.045)
SOC	54715 (0)	8017 (0.101)	53823 (0.060)
SPO	41217 (0)	6100 (0.074)	35748 (0.033)
WOR	35748 (0)	42867 (0.428)	108770 (0.071)

Table 5.1: The number of URLs that were identified as unclassifiable during the pre-processing stage of the topic-classifer. The values in brackets indicate the corresponding amount of retrieved data (in GB).

5.2.4 Transfer of pages between English and foreign crawlers

Classifiable web pages were fed into the *C4 Classifier* for categorization. The classifier had two main components: the *language identifier* and the *topic classifier*. The first component verified if a page was a foreign page. If so, it would categorized the page under WORLD. Otherwise the second component categorized the page under one of the fifteen English categories. To analysis the performance of language identifier, we broke up the entire classified data into two groups: English and Foreign. The English group contains web pages that were retrieved by the English crawlers, and the Foreign group contains those retrieved by WORLD crawler.

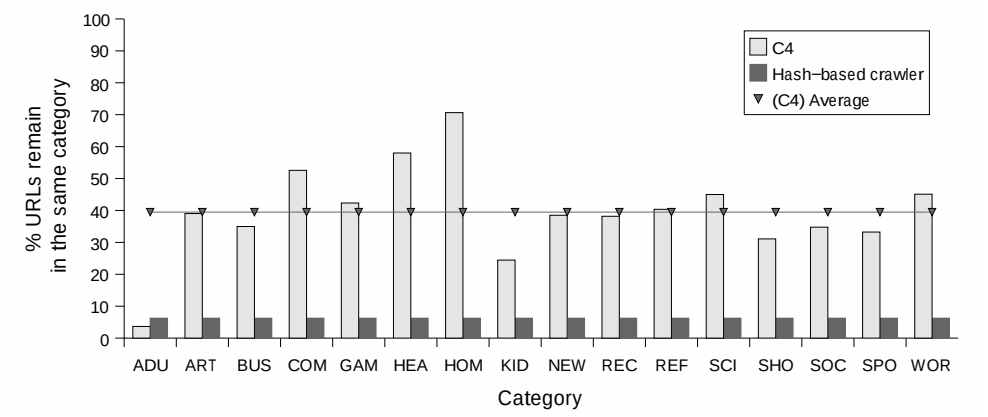

Figure 5.4: Percentage of classifiable URLs assigned to the same category as its parent page. This result is compared against the theoretical result of a hashing approach, which is constant at 6.25% across all categories.

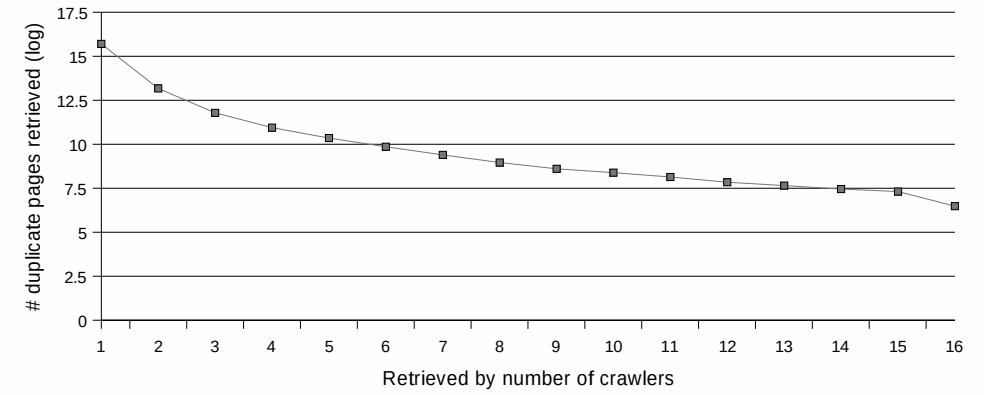

Figure 5.5: Amount of duplicate pages retrieved by multiple crawlers.

Of the 6.801 million classifiable pages, 9.3% were classified as foreign web pages. Among the 0.558 million URLs retrieved by crawler #16 (WORLD), 45% were foreign ones. The probability of an English web page pointing to a foreign one and vice-versa were 6.88% and 55.0%.

5.3 Discussion

5.3.1 Crawlers' retrieval rate

Although the crawlers were ran under numerous network restrictions (such as the daily idling interval and maximum data retrieval rate), they generated 5.6% of the incoming network traffic for the entire university during the month of June 2001. During this time, the crawlers were retrieving data at an average rate of 13.2 KB/sec and none retrieved more than the pre-set limit of 16 KB/sec (see figure 5.6).

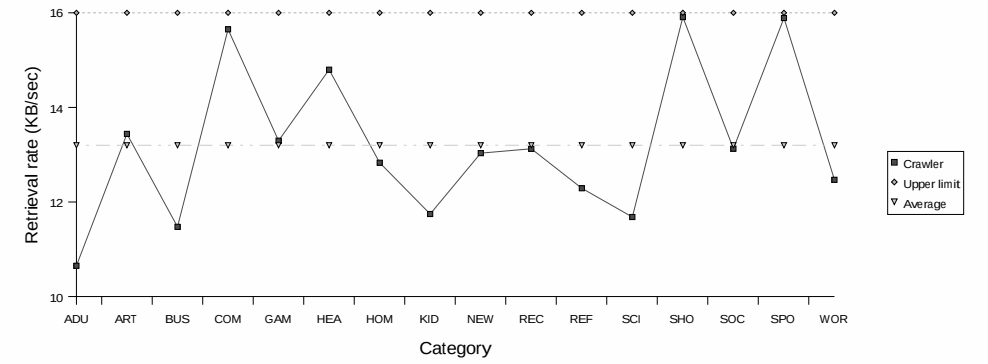

Figure 5.6: The rate of retrieval of each crawler, with an averaged of 13.2 KB/sec.

5.3.2 URL duplication

In this section, we explore the issue of URL duplication in *C4*. The number of pages retrieved by multiple crawlers reflect the effectiveness of partitioning the Web by topic. A badly partitioned Web results in multiple pages visited by multiple crawlers. This redundancy is costly in a distributed web crawling system and should be avoided.

URL duplication occurs when the same URL was found in web pages of different topics. To measure this redundancy, we looked into the URLs of pages that were retrieved by each Multitext crawler. We found that the number of duplicate pages retrieved sharply decreased as the number of crawlers that retrieved them increased (see figure 5.5). 89% of web pages were retrieved by exactly one crawler and more than 96% of web pages were retrieved by two crawlers or less.

Only 654 pages, less than 0.0001% of all pages, were retrieved by every crawlers. Most of these pages did not reflect a single distinctive interest. An example of such pages that are referenced by web pages of all topics is the "Yahoo!" web portal (http://www.yahoo.com).

5.3.3 The influence of unclassifiable pages

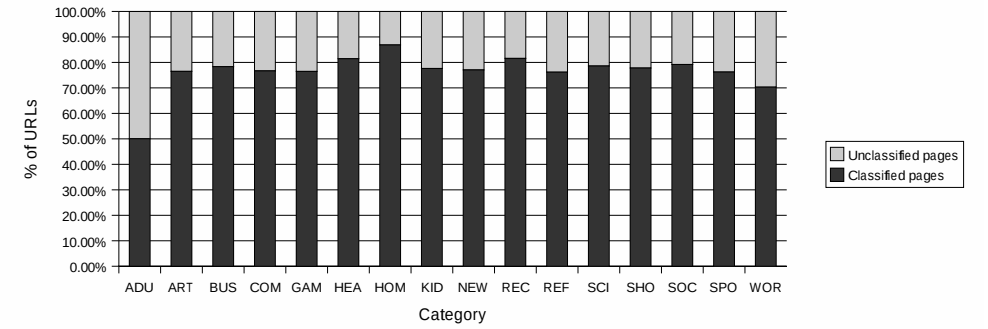

Figure 5.7: Ratio of classifiable pages against the unclassifiable ones.

Next, we examine the number of web pages retrieved by the crawlers that were unclassifiable. If the amount of unclassifable web data is high, then the topic-oriented approach is not effective since most of the data is not classified based on content.

The ratio of classifiable web pages against those that were not was approximately 1 to 4 in all categories except category #0, which was at 1 to 1 (see figure 5.7). This is because pages of Adult content tend to contain graphical pages with no words. In the entire crawl, 23.69% of URLs were unclassifiable. These pages constituted less than 5% of the actual data retrieved. There was a substantial difference between the number of unclassifiable URLs and the size of their content because these web pages were generally small in size:

1. *Empty* pages did not have any HTTP nor HTML content, therefore they contain zero bytes.

2. *Invalid* pages are mostly redirected (3XX) and client error (4XX) pages. These pages are generally small in size.

3. Although it was not always the case, *no-text* pages were usually small in size.

In total, the *no-text* web pages consisted of 4.571 GB of web data, which was 82% of all the unclassifiable web data. These pages made up the most unclassifiable ones because many of them contained large amounts of HTML tags or scripts.

5.3.4 Performance evaluation of the classifier

Next, we assess the performance of *C4 Classifier*. As the amount of unclassifiable web data (5% of all data) is small, this analysis of the classifier does not take it into consideration.

Crawler boundaries A crawler that encounters its boundary frequently is inefficient as it needs to route large numbers of web pages to external crawlers. To

measure the probability of a crawler encountering its boundary, we observe the amount of pages it retrieved that belongs to another crawler. The average probability of a classifiable web page to be assigned the same category as its parent page is 39.48 %. The similar (*hyperlink relationship*) experiment that was carried out earlier in section 4.3.2 yields a probability of 62.4%. The result deteriorates in the crawling experiment because, unlike the earlier experiment which used web pages that were pre-categorized into specific interests, numerous web pages retrieved by the crawlers did not reflect a distinctive topic.

Category #0 classification Crawler #0 retains less than 4% of its child pages (see table 5.4), which is worst than the random approach which yields 6.25%. This result differs greatly from that of *hyperlink relationship* experiment in section 4.3.2 where Category #0 produces the best result. It seems to suggest that either pages of this category often contained links of various topics or the classifier was badly trained in identifying these pages. However, after examining the web pages retrieved by the crawler #0, we found that it was the unclassifiable web pages that caused the mediocre result.

Web pages that contained no text were assigned to the same category (i.e. #0). These pages (e.g. frame pages), although they contained no text, may contain links to other web pages. Therefore, crawler #0 would retrieve these links and categorize them to various topics since their parent pages are from different categories.

English Language Identifier Crawler #16 (WORLD) is assigned to retrieve foreign pages. Its probability of retaining a web page ($\Pr_N^P(\delta)$) is about 45% (see

figure 5.4). Despite that language identifier result had deteriorated by half in comparison to the similar experiment carried out in section 4.4.3, the classifier for crawler #16 still performed better than an average one (at 39.48%). Therefore the language identifier is appropriate for use in *C4*.

5.3.5 The feasibility of topic-oriented partitioning

Despite a decline in the result of *hyperlink relationship* experiment on actual web data, *topic-oriented partitioning* of the Web is still feasible for a distributed web crawling system. If *content-based* hashing were to be used on our distributed crawler, *degree of local page assignment* (i.e. $\Pr_N^P(\delta)$) is $\frac{1}{16} = 6.25\%$. However, *C4* produces a result ($\Pr_{16}^P(\delta)$=39.48%) that is more than six times better than the hashing techniques (see figure 5.4).

Chapter 6

Conclusion

This thesis identifies the importance of web partitioning in distributed web crawling systems and proposes a novel approach by dividing the Web into different topics. The proposed approach employs multiple focused crawlers to retrieve pages from various topics. When a crawler retrieves a page of another topic, it routes the page to the approriate crawler.

Classification is the means to assign web pages to crawlers. Of the topic classification techniques we compared, *Naive Bayes* is the most approriate for our collaborative crawling system. We also propose a simple and efficient, yet effective, language identifier. Despite its simplicity, the identifier was able to obtain an accuracy of over 90%. The language identifier and the topic classifier are then combined to form the *C4 Classifier*, which is incorporated into the crawling system.

C4 is the implementation of our proposed topic-oriented collaborative web crawling system. It divides the Web into sixteen categories, of which fifteen are based on English topics. The system assigns a Multitext crawler to retrieve web

pages from each topic. Although these crawlers operate independently, data are transferred between them constantly.

A crawling experiment was carried out on *C4* over a period of 28 days. The results of the experiment were encouraging. Despite the various restrictions applied to each crawler, the crawling system was able to retrieve data at a rate close to the predefined limit. URL duplication was low; more than 96% of web pages were retrieved by two crawlers or less. Every page retrieved by a crawler has a probability of 39.48% to have its links retrieved by the same crawler. This value is six times higher than the *content-hashing* technique. We view these numbers as strong indications that *topic-oriented collaborative crawling system* is a viable approach to web data gathering.

6.1 Future work

The positive results of the experiment encourage future work to follow up on the results reported in this thesis.

The current implementation of *C4* treats all pages within each topic as having the same qualities. However, consider two web pages δ_1 and δ_2 from the same partition P such that the probabilities of each page to be categorized under P is 60% and 90%. We would like to investigate whether the pages referred to by δ_2 are more likely to be categorized under the same partition than those of δ_1 since δ_2 has a higher probability.

URL duplication is a major concern in *C4*. The retrieval results showed that the percentage of URL duplication is low. Our hypothesis is that as more data

is retrieved, this percentage will continue to decrease until a certain threshold is reached, after which it will increase. We also wonder if URL duplication would increase with finer partitions. Furthur experiments are required to evaluate these hypotheses.

Crawler #0 failed to retain most of its child pages *No-text* pages were assigned to a common partition. This is to ensure that every successful (2XX) pages is assigned uniquely to a web crawler to reduce content duplication across the crawlers. However, the retrieval result of Crawler #0, which is the chosen one, was marred by the *no-text* pages. The overall result may improve if an independent crawler is to retrieve only *no-text*. It was also observed that most of the out-going pages went to Crawler #16 (see appendix A). It is unclear why most of its child pages were identified as foreign ones. Furthur investigations are required.

Bibliography

[1] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1-7):107–117, 1998.

[2] C. Buckley, G. Salton, and J. Allan. The effect of adding relevance information in a relevance feedback environment. In *Proceedings of the 17th Annual International ACM-SIGIR conference on Research and Development in Information Retrieval*, pages 292–301. Springer-Verlag, 1994.

[3] W. Cavnar and J. Trenkle. N-gram-based text categorization. In *Proceedings of 3rd Annual Symposium on Document Analysis and Information Retrieval*, pages 161–175, 1994.

[4] S. Chakrabarti, B. Dom, D. Gibson, J. Kleinberg, P. Raghavan, and S. Rajagopalan. Automatic resource list compilation by analyzing hyperlink structure and associated text. In *Proceedings of the 7th International World Wide Web Conference*, 1998.

[5] Soumen Chakrabarti, Byron E. Dom, and Piotr Indyk. Enhanced hypertext

categorization using hyperlinks. In Laura M. Haas and Ashutosh Tiwary, editors, *Proceedings of SIGMOD-98, ACM International Conference on Management of Data*, pages 307–318, Seattle, US, 1998. ACM Press, New York, US.

[6] Soumen Chakrabarti, David Gibson, and Kevin S. McCurley. Surfing the web backwards. *WWW8 / Computer Networks*, 31(11-16):1679–1693, 1999.

[7] Soumen Chakrabarti, Martin van den Berg, and Byron Dom. Focused crawling: A new approach to topic-specific web resource discovery. In *8th World Wide Web Conference*, Toronto, May 1999.

[8] Junghoo Cho. *Crawling the Web: Discovery and Maintenance of Large-scale Web Data*. PhD thesis, Stanford University, 2001.

[9] Junghoo Cho, Hector García-Molina, and Lawrence Page. Efficient crawling through URL ordering. *Computer Networks and ISDN Systems*, 30(1-7):161–172, 1998.

[10] Charles L. Clarke, Gordon V. Cormack, and Elizabeth A. Tudhope. Relevance ranking for one- to three-term queries. In *Proceedings of RIAO-97, 5th International Conference "Recherche d'Information Assistee par Ordinateur"*, pages 388–400, Montreal, CA, 1997.

[11] Jim Cowie, Yevgeny Ludovik, and Ron Zacharski. An autonomous, web-based, multilingual corpus collection tool. In *International Conference on Natural Language Processing and Industrial Applications*, 1998.

[12] Altigran Soares da Silva, Eveline A. Veloso, Paulo Braz Golgher, Berthier A. Ribeiro-Neto, Alberto H. F. Laender, and Nivio Ziviani. Cobweb - a crawler for the Brazilian Web. In *SPIRE/CRIWG*, pages 184–191, 1999.

[13] Brian D. Davison. Topical locality in the web. In *Research and Development in Information Retrieval*, pages 272–279, 2000.

[14] Jeffrey Dean and Monika Rauch Henzinger. Finding Related Pages in the World Wide Web. In *Proceedings of the 8th International World Wide Web (WWW) Conference*, volume 31, pages 1467–1479, 1999.

[15] Scott C. Deerwester, Susan T. Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990.

[16] Michelangelo Diligenti, Frans Coetzee, Steve Lawrence, C. Lee Giles, and Marco Gori. Focused crawling using context graphs. In *26th International Conference on Very Large Databases, VLDB 2000*, pages 527–534, Cairo, Egypt, 10-14 September 2000.

[17] Susan T. Dumais and Hao Chen. Hierarchical classification of web content. In Nicholas J. Belkin, Peter Ingwersen, and Mun-Kew Leong, editors, *Proceedings of SIGIR-00, 23rd ACM International Conference on Research and Development in Information Retrieval*, pages 256–263, Athens, GR, 2000. ACM Press, New York, US.

[18] Joel L Fagan. *Experiments in Automatic Phrase Indexing For Document Re-*

trieval: A Comparison of Syntactic and Non-Syntactic Methods. PhD thesis, Cornell University, 1987.

[19] J. Fiedler and J. Hammer. Using the Web efficiently: Mobile Crawlers. Technical report, University of Florida, Gainesville, 1998.

[20] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. Technical report, RFC-2616, 1999.

[21] Norbert Fuhr. Models for retrieval with probabilistic indexing. *Information Processing and Management*, 25(1):55–72, 1989.

[22] S. Gauch, G. Wang, and M. Gomez. ProFusion: Intelligent fusion from multiple, distributed search engines. *JUCS: Journal of Universal Computer Science*, 2(9):637–649, 1996.

[23] David Gibson, Jon M. Kleinberg, and Prabhakar Raghavan. Inferring web communities from link topology. In *UK Conference on Hypertext*, pages 225–234, 1998.

[24] David Hawking, Nick Craswell, Paul B. Thistlewaite, and Donna Harman. Results and challenges in web search evaluation. *WWW8 / Computer Networks*, 31(11-16):1321–1330, 1999.

[25] Monika Rauch Henzinger, Allan Heydon, Michael Mitzenmacher, and Marc Najork. Measuring index quality using random walks on the Web. *WWW8 / Computer Networks*, 31(11-16):1291–1303, 1999.

[26] W. Hersh, C. Buckley, T. Leone, and D. Hickam. Ohsumed: An interactive retrieval evaluation and new large test collection for research. In *Proceedings of the 17th Annual ACM SIGIR Conference*, pages 192–201, 1994.

[27] J. Howard, M. Kazar, S. Menees, D. Nichols, M. Satyanarayanan, R. Sidebotham, and M. West. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems*, 6(1):51–81, February 1988.

[28] Adele E. Howe and Daniel Dreilinger. SAVVYSEARCH: A metasearch engine that learns which search engines to query. *AI Magazine*, 18(2):19–25, 1997.

[29] M. James. *Classification Algorithm*. New York: John Wiley & Son, 1985.

[30] Thorsten Joachims. A probabilistic analysis of the Rocchio algorithm with TFIDF for text categorization. In *Proceedings of the 14th International Conference on Machine Learning*, pages 143–151. Morgan Kaufmann, 1997.

[31] Thorsten Joachims. Text categorization with support vector machines: learning with many relevant features. In Claire Nédellec and Céline Rouveirol, editors, *Proceedings of ECML-98, 10th European Conference on Machine Learning*, pages 137–142, Chemnitz, DE, 1998. Springer Verlag, Heidelberg, DE.

[32] Karen Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28:11–21, 1972.

[33] Karen Sparck Jones. Collection properties influencing automatic term classification performance. *Information Storage and Retrieval*, 9:499–513, 1973.

[34] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999.

[35] D. Koller and M. Sahami. Hierachically classifying documents using very few words. In *Machine Learning: Proceedings of the Fourteenth Internationa Conference*, 1997.

[36] M. Koster. Robots in the Web: threat or treat. *ConneXions*, 9(4), 1995.

[37] S. Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew Tomkins. Trawling the web for emerging cyber-communities. *WWW8 / Computer Networks*, 31(11-16):1481–1493, 1999.

[38] Steve Lawrence and C. Lee Giles. Searching the World Wide Web. *Science*, 280(5360):98–100, 1998.

[39] Steve Lawrence and C. Lee Giles. Accessibility of information on the web. *Nature*, 400:107–109, 1999.

[40] D. Lewis. An evaluation of phrasal and clustered representations on a text categorization task. In *Fifteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 37–50, 1992.

[41] D. Lewis. Feature Selection and Feature Extraction for Text Categorization. In *Proceedings of Speech and Natural Language Workshop*, pages 212–217, San Mateo, California, 1992. Morgan Kaufmann.

[42] D. Lewis and W. Bruce Croft. Term clustering of syntactic phrases. In *Research and Development in Information Retrieval*, pages 385–404, 1990.

[43] L. Masinter, H. Alvestrand, Pirsenteret Maxware, D. Zigmond, and R. Petke. Guidelines for new URL Schemes. Technical report, RFC-2718, 1999.

[44] Tom Mitchell. *Machine Learning*. McGraw Hill, 1997.

[45] Matthew Montebello. WebSearch:Optimizing Recall/Precision scores in IR over the WWW. In *Proceedings of the SIGIR '98*, 1998.

[46] J. Murai, M. Crispin, and E. van der Poel. Japanese character encoding for internet messages. Technical report, RFC-1468, 1993.

[47] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical report, Stanford University, 1998.

[48] J. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. Technical report, Microsoft Research, Redmond, Washington, 1998.

[49] Yonggang Qiu and Hans-Peter Frei. Concept-based query expansion. In *Proceedings of SIGIR-93, 16th ACM International Conference on Research and Development in Information Retrieval*, pages 160–169, Pittsburgh, US, 1993.

[50] Ronald L. Rivest. The MD5 Message-Digest Algorithm. Technical report, RFC-1321, 1992.

[51] J. Rocchio. Relevance feedback in information retrieval. In *The SMART Retrieval System– Experiments in Automatic Document Processing*, pages 313–323. Prentice Hall, 1971.

[52] G. Salton. *Automatic text processing, the transformation analysis and retrieval of information by computer*. Addison - Wesley, 1989.

[53] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing and Management*, 24(5):513–523, 1988.

[54] Gerard Salton and C.S. Yang. On the specification of term values in automatic indexing. *Journal of Documentation*, 29:351–372, April 1973.

[55] S. Salzberg. Improving classification methods via feature selection. Technical report, Johns Hopkins University, 1992.

[56] E. Selberg and O. Etzioni. Multi-service search and comparison using the MetaCrawler. In *Proceedings of the 4th International World Wide Web (WWW) Conference*, 1995.

[57] D. Souter, G. Churcher, J. Hayes, J. Hughes, and S. Johnson. Natural language identification using corpus-based models. *Journal of Linguistics*, 13:183–203, 1994.

[58] Raymie Stata, Krishna Bharat, and Farzin Maghoul. The term vector database: fast access to indexing terms for web pages. In *Proceedings of the 9th International World Wide Web (WWW9) Conference*, pages 247–256, May 2000.

[59] Piotr Indyk Taher H. Haveliwala, Aristides Gionis. Scalable techniques for clustering the web. In *Proceedings of the WebDB*, 2000.

[60] Shang-Hua Teng, Qi Lu, M.Eichstaedt D. ford, and T. Lehman. Collaborative web crawling: Information gathering/processing over internet. In *32nd Hawaii International Conference on System Sciences*, 1999.

[61] T. Tokunaga and M. Iwayama. Text categorization based on weighted inverse document frequency. Technical report, Tokyo Institute of Technology, 1994.

[62] E.M. Voorhees. Query Expansion Using Lexical-Semantic Relations. In *Proceedings of the 17th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval*, Dublin, Ireland, July 1994.

[63] Jinxi Xu and W. Bruce Croft. Query expansion using local and global document analysis. In *Research and Development in Information Retrieval*, pages 4–11, 1996.

[64] Hayato Yamana, Kent Tamura, Hiroyuki Kawano, Satoshi Kamei, Masanori Harada, Hideki Nishimura, Isao Asai, Hiroyuki Kusumoto, Yoichi Shinoda, and Yoichi Muraoka. Experiments of collecting WWW information using distributed WWW robots. In *Research and Development in Information Retrieval*, pages 379–380, 1998.

[65] Y. Yang and X. Liu. A re-examination of text categorization methods. In *22nd Annual International SIGIR*, pages 42–49, Berkley, August 1999.

[66] Yiming Yang and C.G. Chute. An example-based mapping method for text classification and retrieval. *ACM Transactions on Information Systems (TOIS)*, 1994.

[67] Yiming Yang and Jan O. Pedersen. A comparative study on feature selection in text categorization. In *Proceedings of the 14th International Conference on Machine Learning*, pages 412–420. Morgan Kaufmann, 1997.

[68] Yiming Yang and John W. Wilbur. Using corpus statistics to remove redundant words in text categorization. *Journal of the American Society for Information Science*, 47(5):357–369, 1996.

[69] F. Yergeau, G. Nicol, G. Adams, and M. Durst. Internationalization of the HyperText Markup Language. Technical report, RFC-2070, 1997.

[70] Oren Zamir, Oren Etzioni, Omid Madani, and Richard M. Karp. Fast and intuitive clustering of web documents. In *3rd International Conference on Knowledge Discovery ad Data Mining*, pages 287–290, 1997.

Appendix A

URLs distribution to crawlers

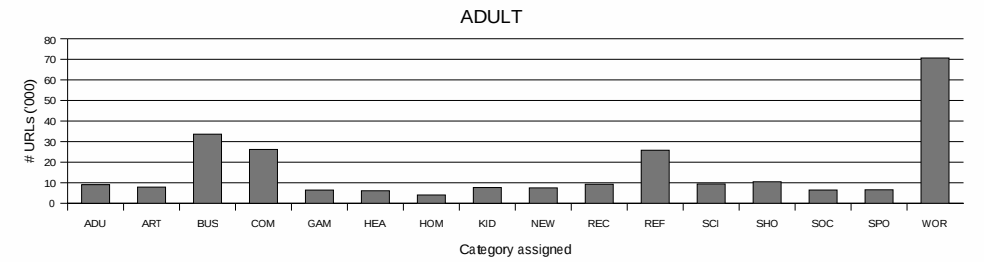

Figure A.1: Distribution of URLS from *Adult* crawler to others.

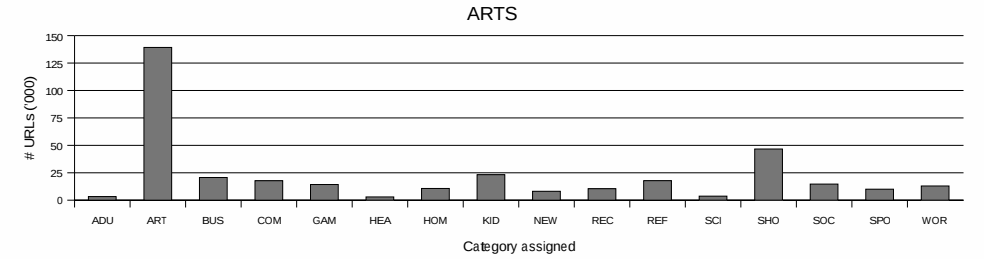

Figure A.2: Distribution of URLS from *Arts* crawler to others.

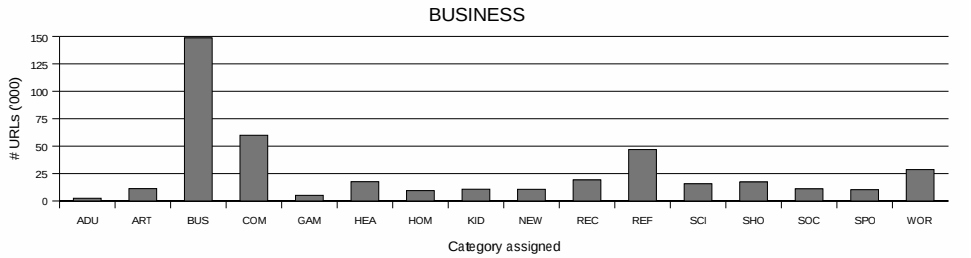

Figure A.3: Distribution of URLS from *Business* crawler to others.

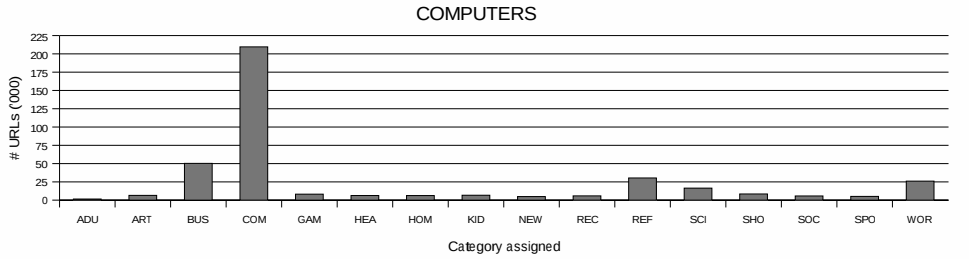

Figure A.4: Distribution of URLS from *Computers* crawler to others.

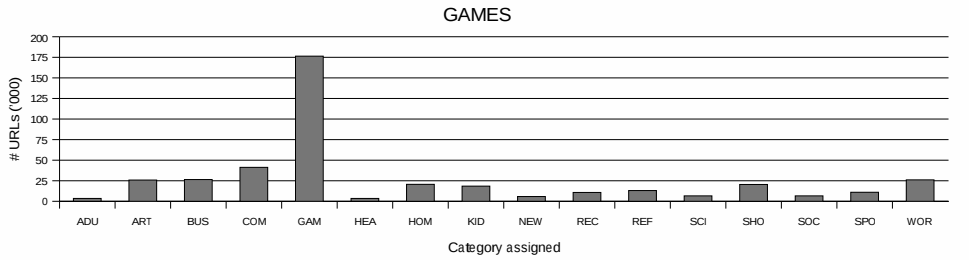

Figure A.5: Distribution of URLS from *Games* crawler to others.

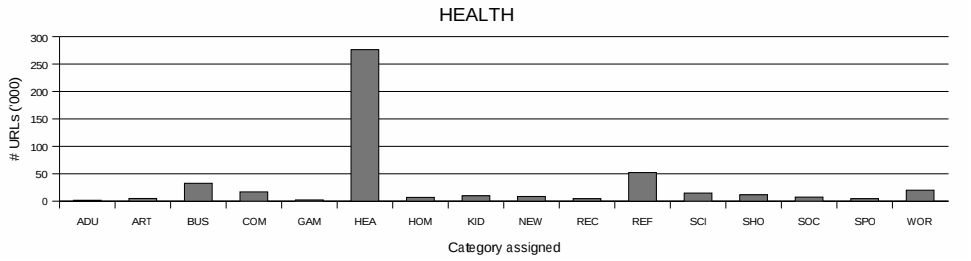

Figure A.6: Distribution of URLS from *Health* crawler to others.

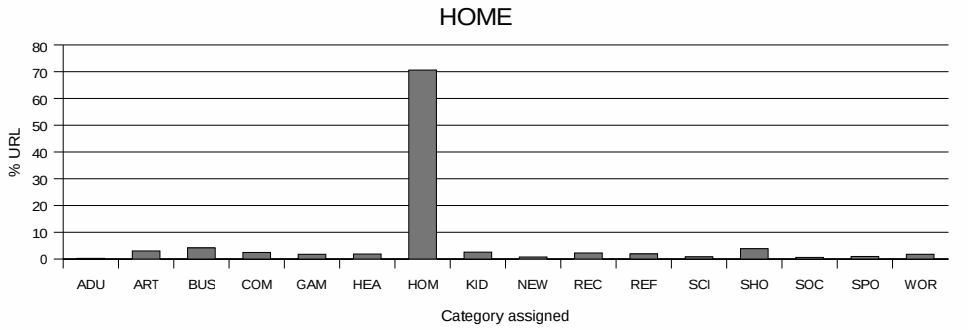

Figure A.7: Distribution of URLS from *Home* crawler to others.

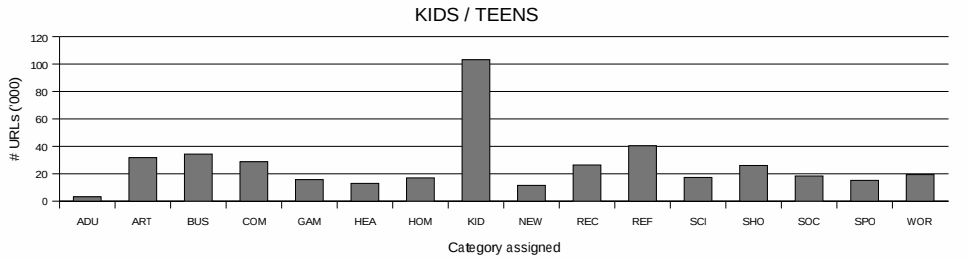

Figure A.8: Distribution of URLS from *Kids/Teens* crawler to others.

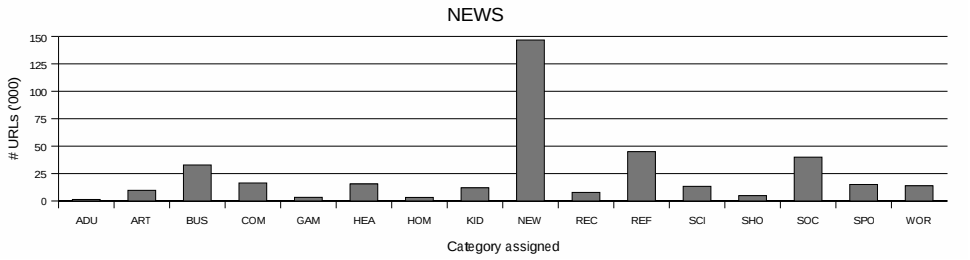

Figure A.9: Distribution of URLS from *News* crawler to others.

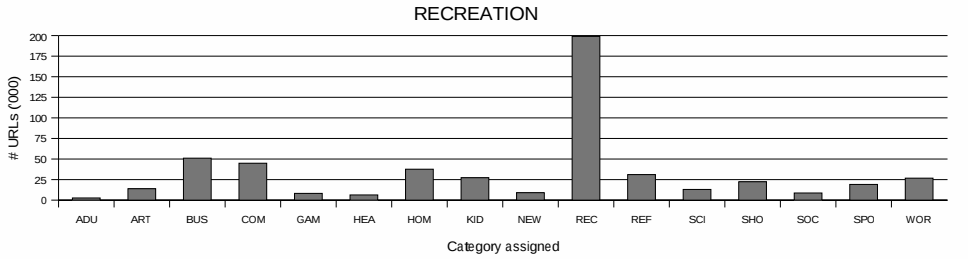

Figure A.10: Distribution of URLS from *Recreation* crawler to others.

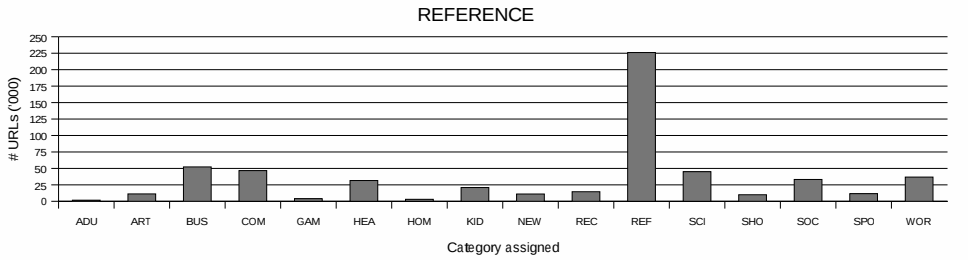

Figure A.11: Distribution of URLS from *Reference* crawler to others.

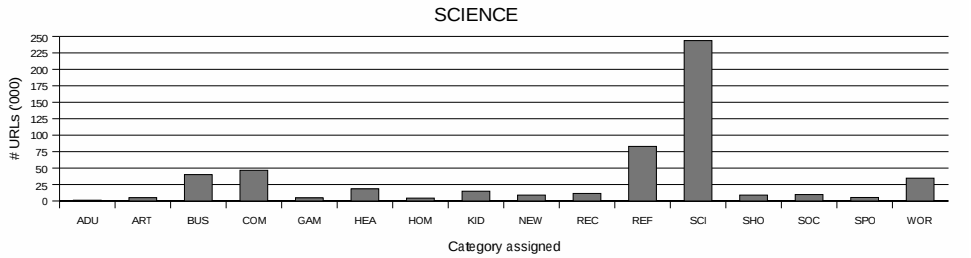

Figure A.12: Distribution of URLS from *Science* crawler to others.

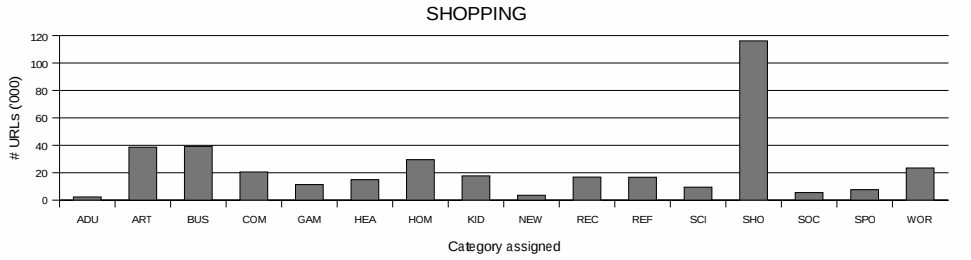

Figure A.13: Distribution of URLS from *Shopping* crawler to others.

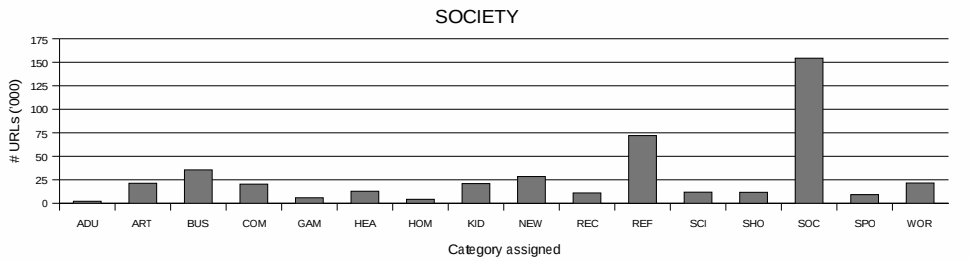

Figure A.14: Distribution of URLS from *Society* crawler to others.

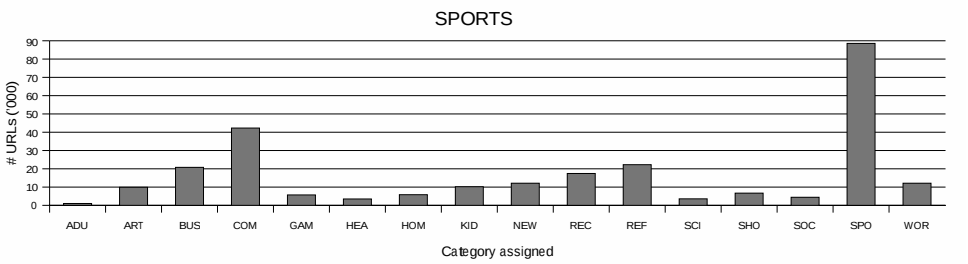

Figure A.15: Distribution of URLS from *Sports* crawler to others.

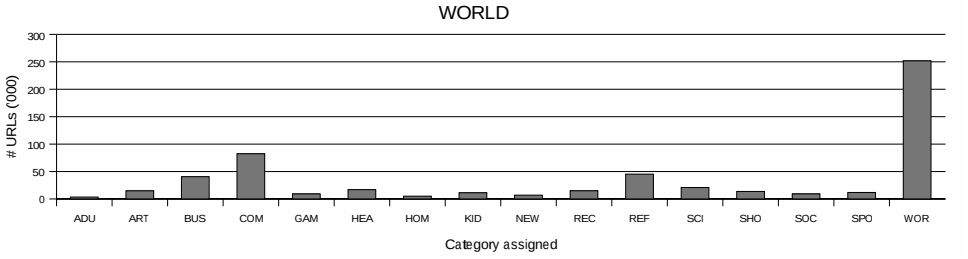

Figure A.16: Distribution of URLS from *World* crawler to others.

Appendix B

Data distribution to crawlers

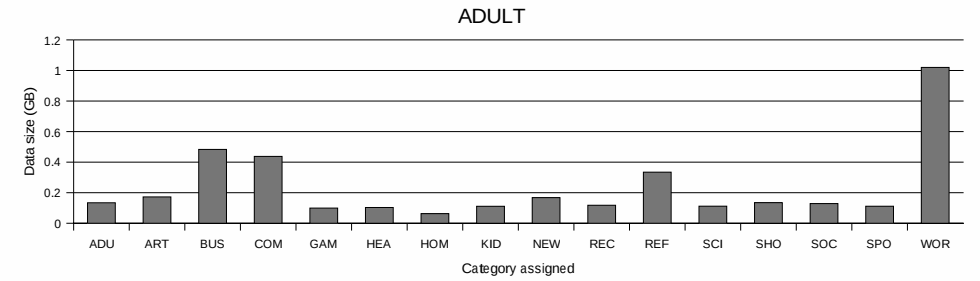

Figure B.1: Amount of data distributed from *Adult* crawler to others.

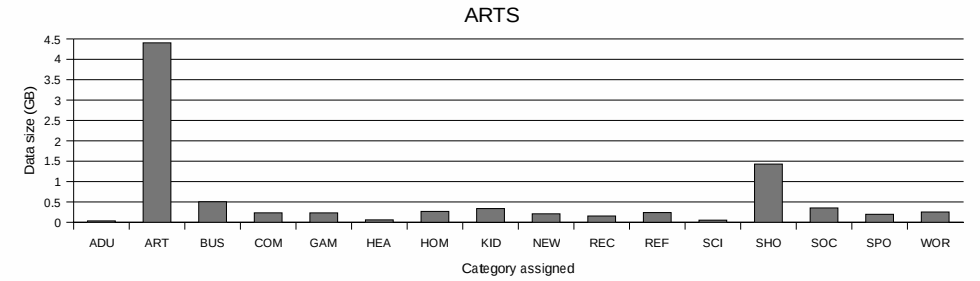

Figure B.2: Amount of data distributed from *Artx* crawler to others.

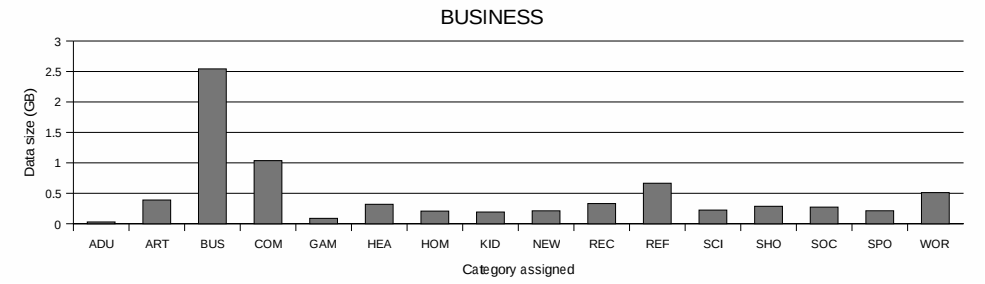

Figure B.3: Amount of data distributed from *Business* crawler to others.

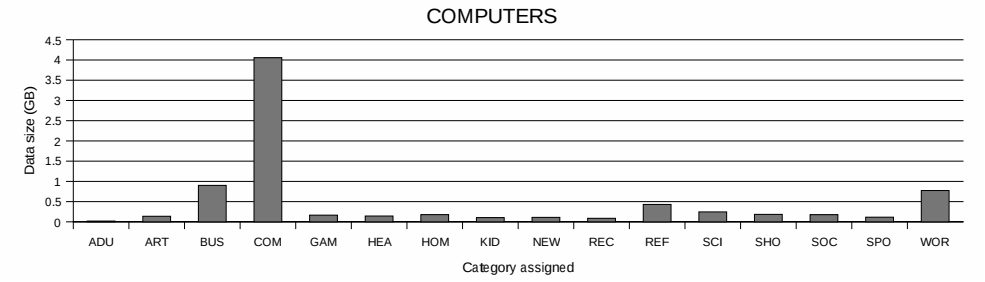

Figure B.4: Amount of data distributed from *Computers* crawler to others.

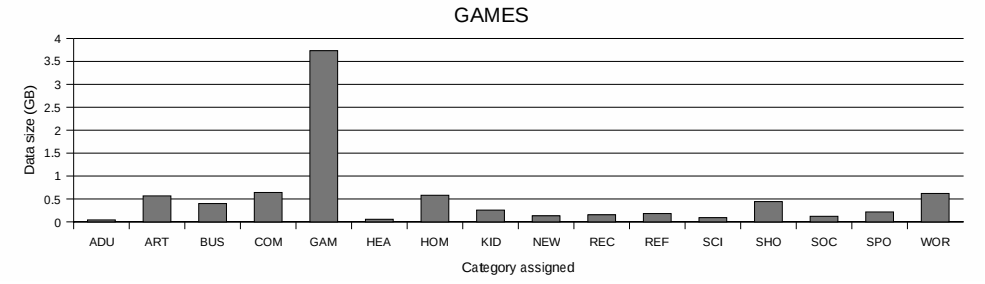

Figure B.5: Amount of data distributed from *Games* crawler to others.

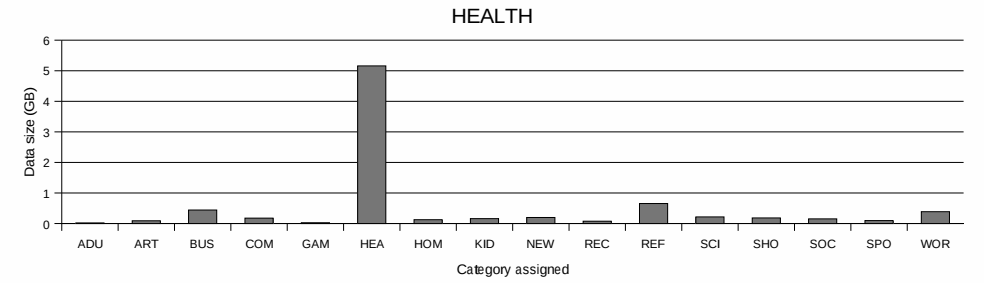

Figure B.6: Amount of data distributed from *Health* crawler to others.

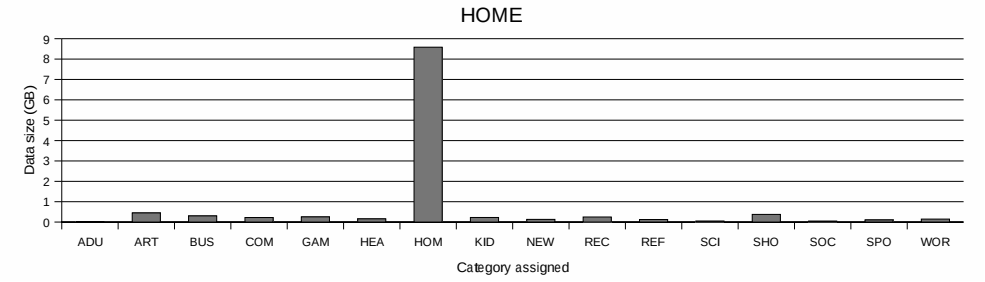

Figure B.7: Amount of data distributed from *Home* crawler to others.

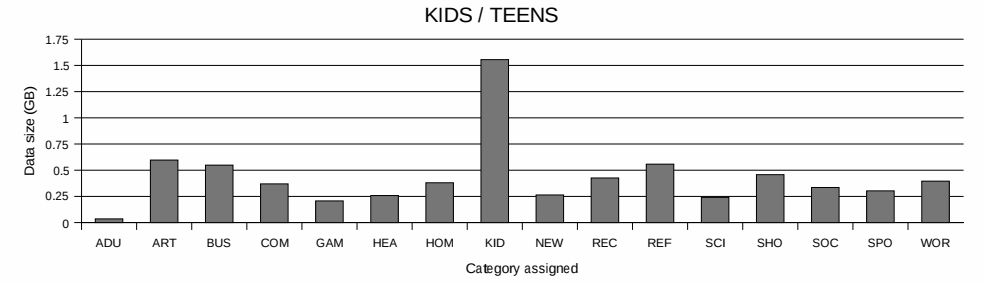

Figure B.8: Amount of data distributed from *Kids/Teens* crawler to others.

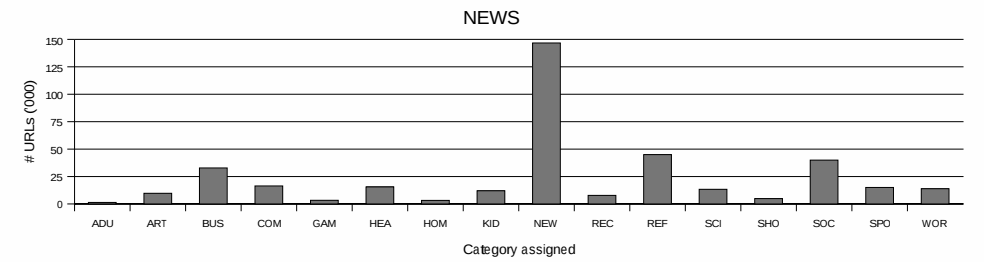

Figure B.9: Amount of data distributed from *News* crawler to others.

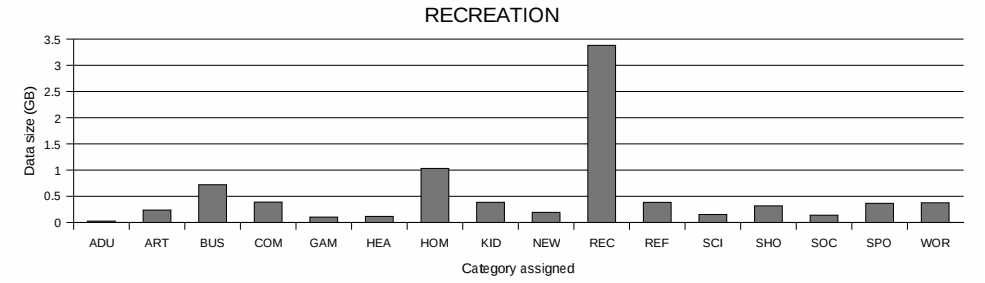

Figure B.10: Amount of data distributed from *Recreation* crawler to others.

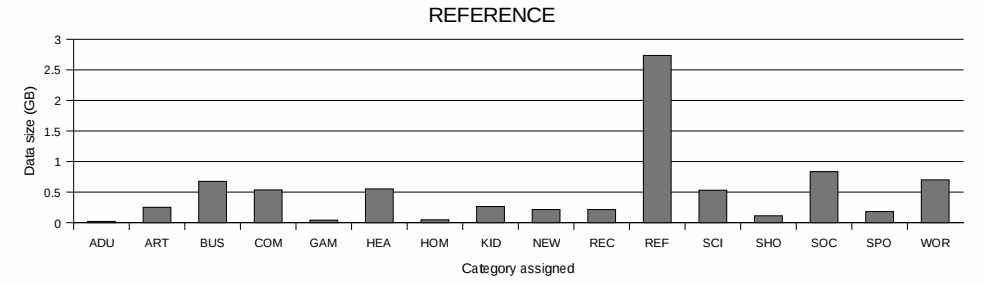

Figure B.11: Amount of data distributed from *Reference* crawler to others.

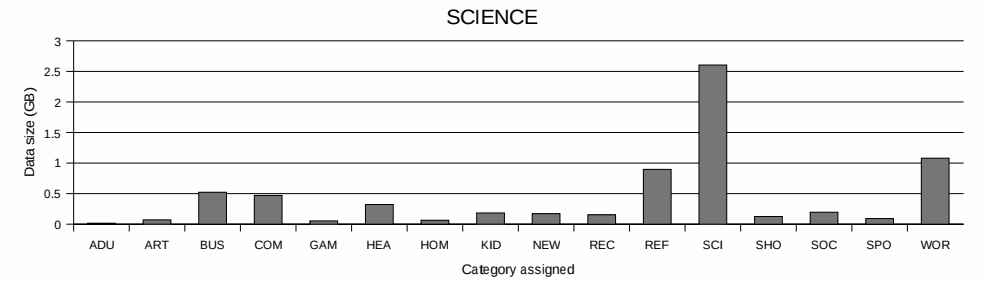

Figure B.12: Amount of data distributed from *Science* crawler to others.

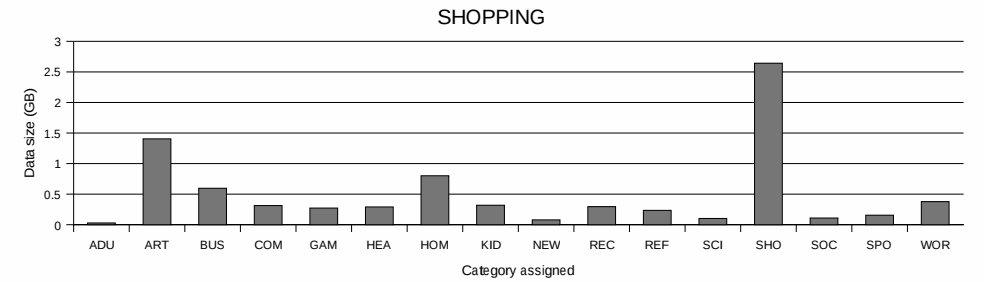

Figure B.13: Amount of data distributed from *Shopping* crawler to others.

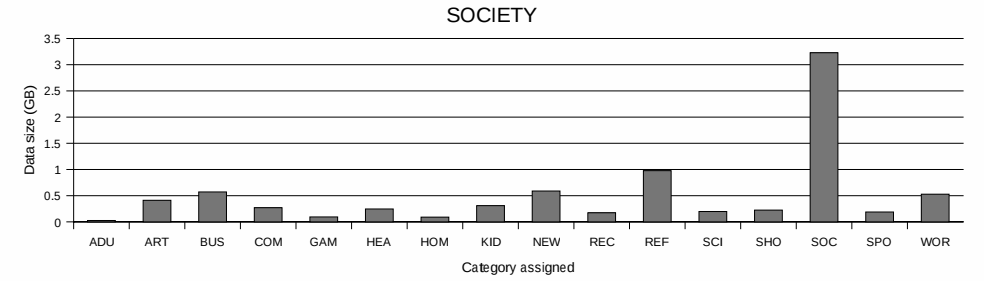

Figure B.14: Amount of data distributed from *Society* crawler to others.

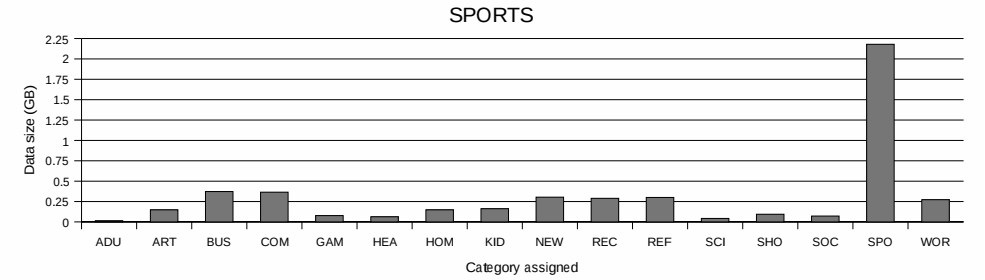

Figure B.15: Amount of data distributed from *Sports* crawler to others.

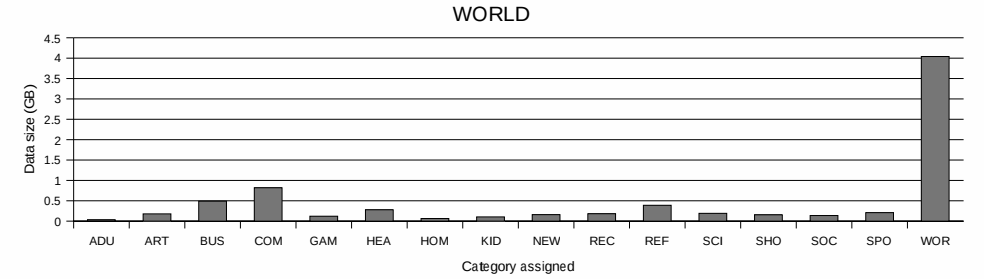

Figure B.16: Amount of data distributed from *World* crawler to others.

Appendix C

Percentage of distribution to crawlers

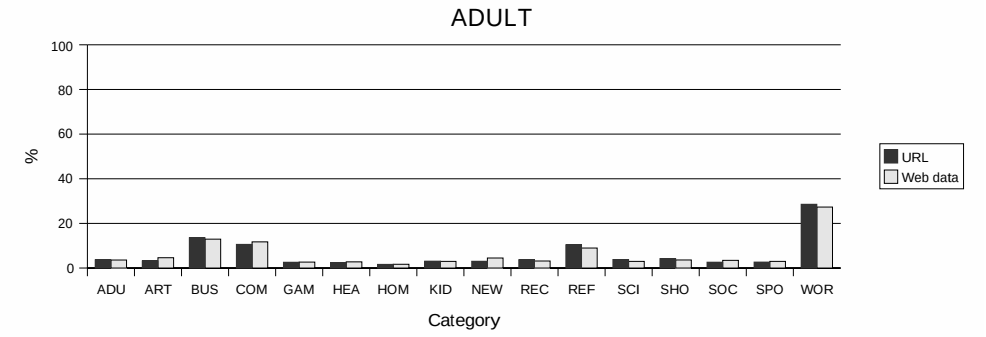

Figure C.1: Percentage of distribution (of URLs and Web data) from *Adult* crawler to others.

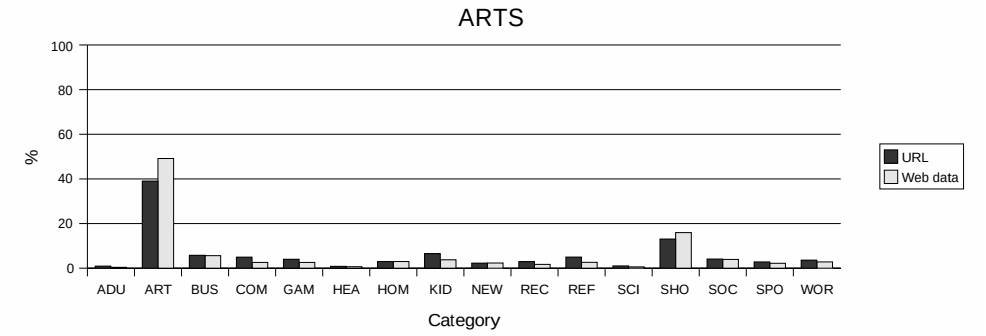

Figure C.2: Percentage of distribution (of URLs and Web data) from *Arts* crawler to others.

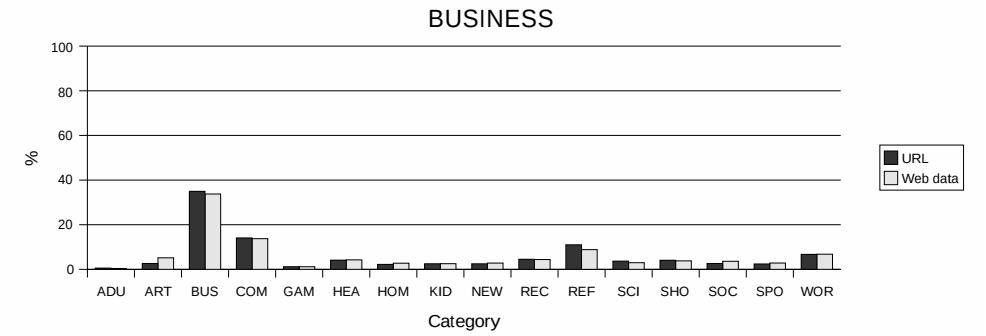

Figure C.3: Percentage of distribution (of URLs and Web data) from *Business* crawler to others.

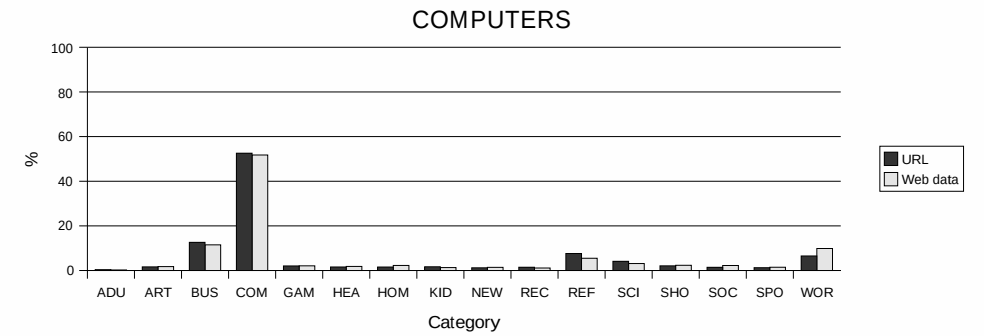

Figure C.4: Percentage of distribution (of URLs and Web data) from *Computers* crawler to others.

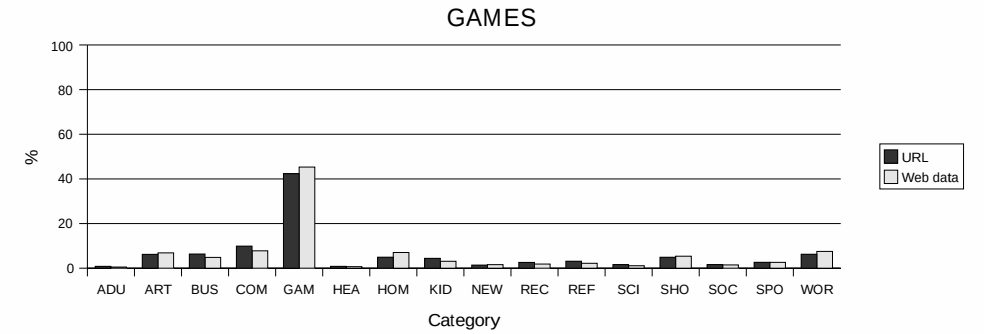

Figure C.5: Percentage of distribution (of URLs and Web data) from *Games* crawler to others.

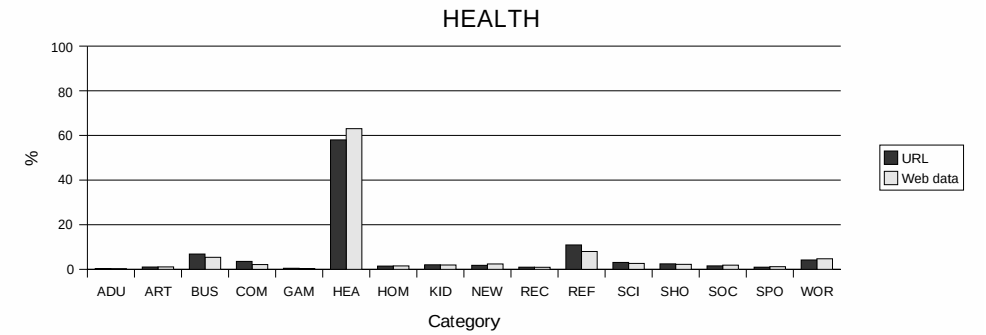

Figure C.6: Percentage of distribution (of URLs and Web data) from *Health* crawler to others.

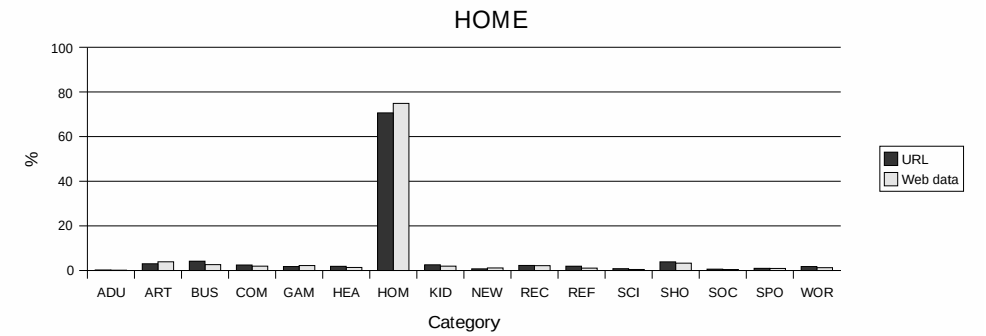

Figure C.7: Percentage of distribution (of URLs and Web data) from *Home* crawler to others.

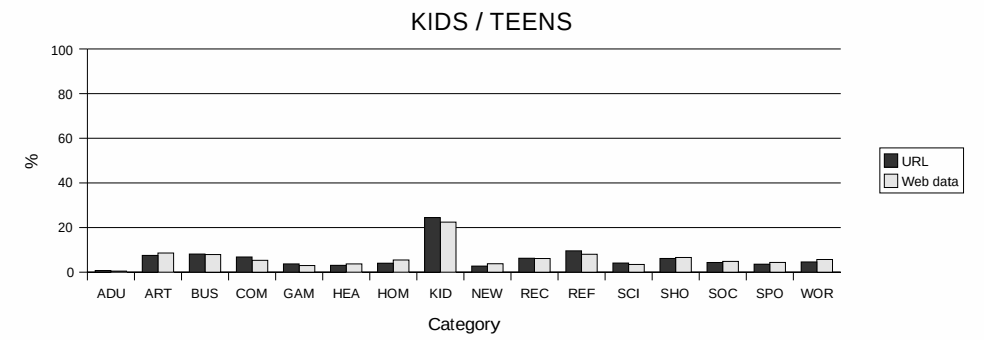

Figure C.8: Percentage of distribution (of URLs and Web data) from *Kids/Teens* crawler to others.

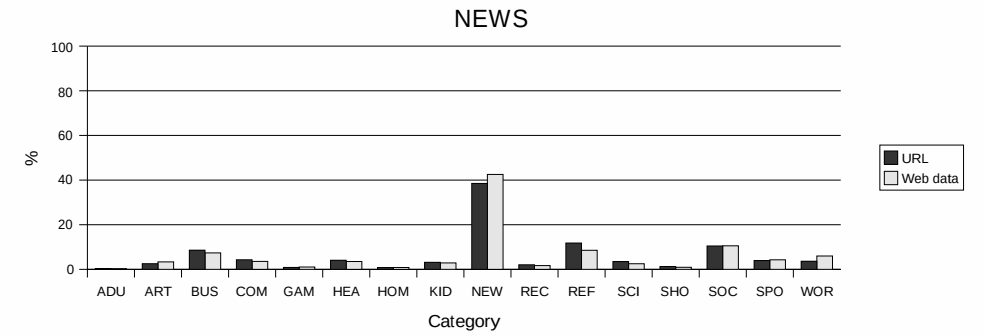

Figure C.9: Percentage of distribution (of URLs and Web data) from *News* crawler to others.

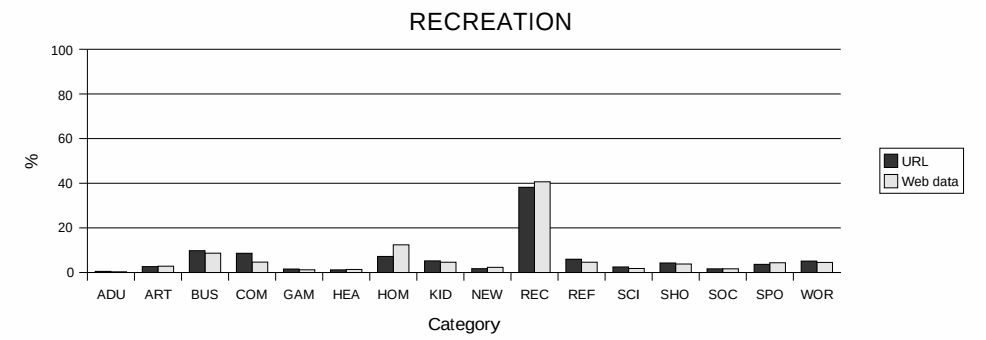

Figure C.10: Percentage of distribution (of URLs and Web data) from *Recreation* crawler to others.

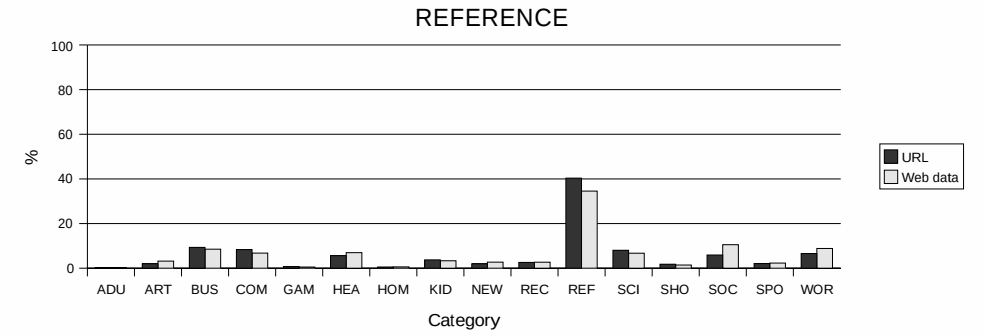

Figure C.11: Percentage of distribution (of URLs and Web data) from *Reference* crawler to others.

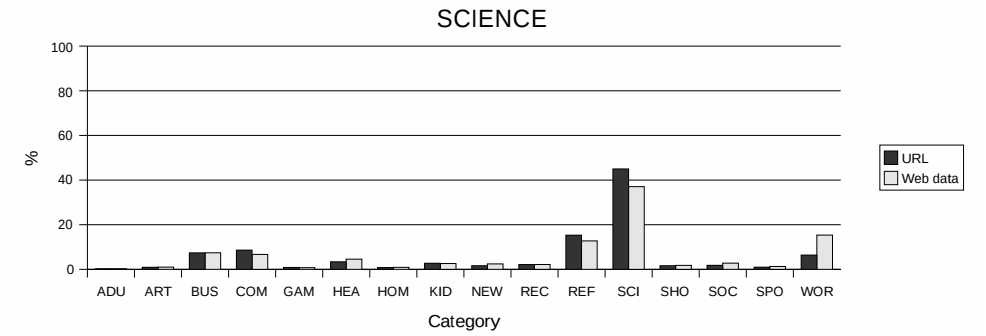

Figure C.12: Percentage of distribution (of URLs and Web data) from *Science* crawler to others.

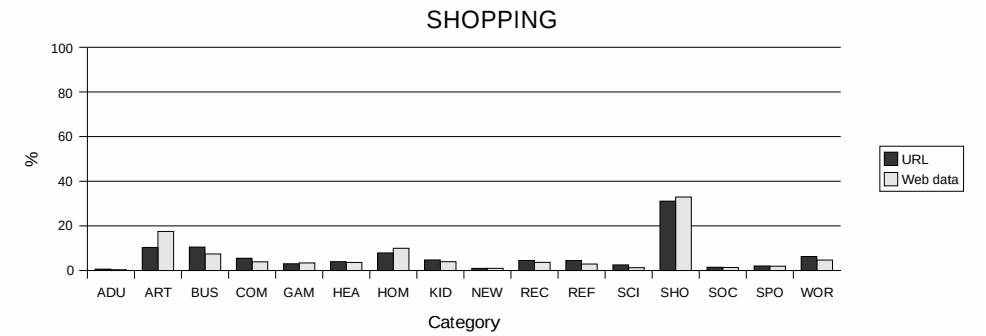

Figure C.13: Percentage of distribution (of URLs and Web data) from *Shopping* crawler to others.

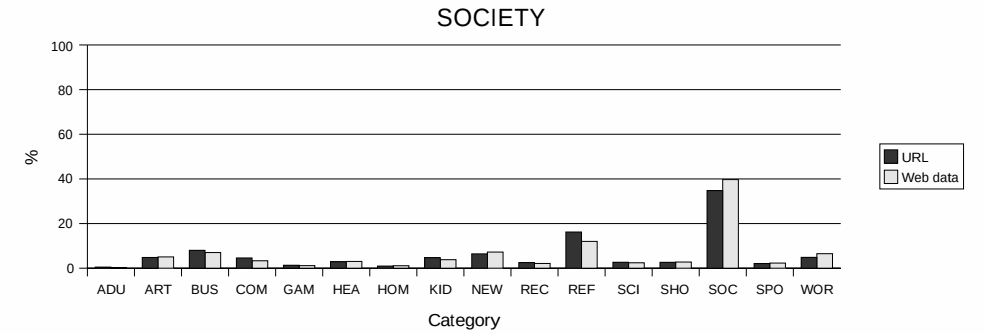

Figure C.14: Percentage of distribution (of URLs and Web data) from *Society* crawler to others.

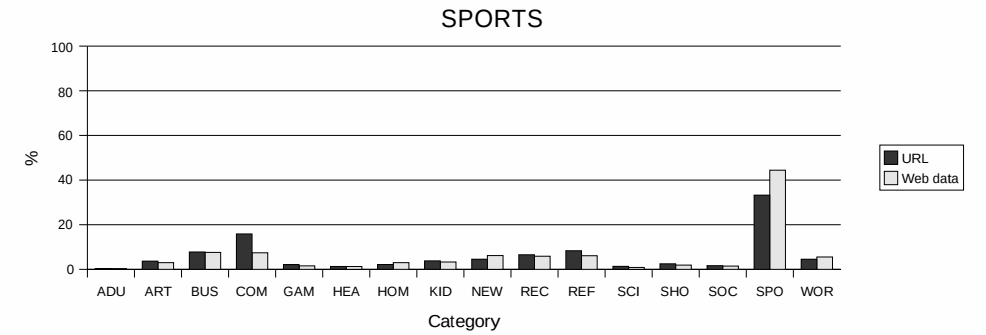

Figure C.15: Percentage of distribution (of URLs and Web data) from *Sports* crawler to others.

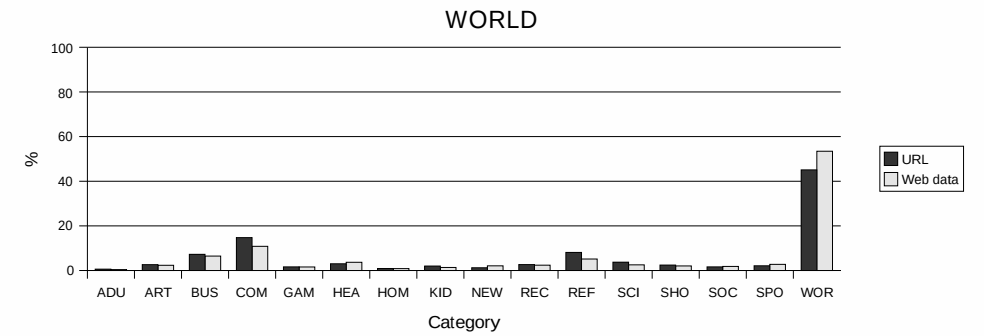

Figure C.16: Percentage of distribution (of URLs and Web data) from *World* crawler to others.