Effective Web Crawlers

A thesis submitted for the degree of
Doctor of Philosophy

Halil Ali B.App.Sc (Hons.),
School of Computer Science and Information Technology,
Science, Engineering, and Technology Portfolio,
RMIT University,
Melbourne, Victoria, Australia.

17th March, 2008

Declaration

I certify that except where due acknowledgement has been made, the work is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program; and, any editorial work, paid or unpaid, carried out by a third-party is acknowledged.

Halil Ali
School of Computer Science and Information Technology
RMIT University
17th March, 2008

Acknowledgments

First and foremost, I would like to offer my deepest gratitude to my supervisors, Andrew Turpin and Justin Zobel, as well as my former supervisor Hugh Williams. Without their continued support, invaluable advice and patience, this thesis would not have been possible.

In addition, I would like to thank my SEG colleagues who have put up with my ramblings and delusions of grandeur, while all the while helping me by providing feedback and assistance. In particular, I wish to thank Steven "Super Steeeve" Garcia, Abhijit Chattaraj, Bodo von Billerbeck, Falk Scholer, Milad Shokouhi, and Nick Lester for their technical assistance and friendship. In addition, I would like to thank Jovan Pehcevski, Nikolas Askitis, Pauline Chou, Ranjan Sinha, Timo Volkmer, Vaughan Shanks, Yaniv Bernstein, Yohannes Tsegay, Jelita Asian and any others I may have missed. Thank you all for making the SEG cave and Tardis such a warm environment to work in. I would like to thank all the support staff that have helped maintain the machines and environments in which I worked.

Thank you to my close friends, James Baylis, Peter Owen, and Peter Pavlovic for your continued support and friendship. Finally, I would like to thank my parents, Feriha and Erdal, as well as my siblings Ali, Kerim, and Soyda. You have all supported my aspirations and providing a nurturing environment for my work to thrive. I thank you all from the bottom of my heart.

Credits

Portions of the material in this thesis have previously appeared in the following publications:

- What's Changed? Measuring Document Change in Web Crawling for Search Engines. In Proceedings of the 10th International Symposium on String Processing and Information Retrieval (SPIRE) 2003 Manaus, Brazil, October 2003.

This work was supported by the Australian Research Council and the Search Engine Group at RMIT University.

The thesis was written in WinEdt on Windows XP, and typeset using the LaTeX 2_ε document preparation system.

Note

Unless otherwise stated, all fractional results have been rounded to the displayed number of decimal figures.

Contents

List of Figures

List of Tables

List of Algorithms

Abstract

Web crawlers are the component of a search engine that must traverse the Web, gathering documents in a local repository for indexing by a search engine so that they can be ranked by their relevance to user queries. As with any system that replicates data in an autonomously updated environment, there are issues with maintaining up-to-date copies of documents.

When documents are retrieved by a crawler and have subsequently been altered on the Web, the effect is an inconsistency in user search results. While the impact depends on the type and volume of change, many existing algorithms do not take the degree of change into consideration, instead using simple measures that consider any change as significant. Furthermore, many crawler evaluation metrics do not consider index freshness or the amount of impact that crawling algorithms have on user results. Most of the existing work makes assumptions about the change rate of documents on the Web, or relies on the availability of a long history of change.

In this thesis, we investigate five specific approaches to improving index consistency: detecting meaningful change, measuring the impact of a crawl on collection freshness from a user perspective, developing a framework for evaluating crawler performance, determining the effectiveness of various stateless crawl ordering schemes, and proposing and evaluating the effectiveness of a dynamic crawl approach. Our work is concerned specifically with cases where there is little or no past change statistics with which predictions can be made.

In our work we analyse different measures of change and introduce a novel approach to measuring the impact of recrawl schemes on search engine users. We show that our Words scheme is an effective metric that detects important changes that affect user results. Other well-known schemes, such as the widely used shingling approach, have to retrieve around twice the volume of data to achieve the same level of effectiveness as the Words scheme.

Furthermore, while many studies have assumed that the Web changes according to a model, our experimental results are all based on real web documents. In our work we also highlight the development of the testbed of documents we use for testing our schemes.

We analyse twelve stateless crawl ordering schemes that have no past change statistics with which to predict which documents will change. In particular we compare the INLINK, OUTLINK, HUB, URLLENGTH, URLDEPTH, ALPHA, QUERY, LARGE, RANDOM, BREADTH, DEPTH, and PAGERANK schemes. While some of these schemes have been tested and shown to be effective, none, to our knowledge, has been tested to determine effectiveness in crawling changed documents. We empirically show that the effectiveness of these various schemes depends on the topology and dynamics of the domain crawled. For the ABC domain we find that the HUB scheme is most effective, while the URLLENGTH scheme works best for the more volatile CSIRO domain. Our results indicate that no one static crawl ordering scheme can effectively maintain freshness, motivating our work on dynamic approaches. In addition we evaluate the utility of a popularity-based evaluation system, finding that it is also ineffective.

We then present our own novel approach to maintaining freshness, which uses the anchor text linking documents to determine the likelihood of a document changing, based on statistics gathered during the current crawl. We show that this scheme is highly effective when combined with existing stateless crawl ordering schemes. Our approach allows the crawling scheme to improve both freshness and quality of a collection by combining it with a scheme such as PageRank. Our scheme improves freshness regardless of which stateless scheme it is used in conjunction with, since it uses both positive and negative reinforcement to determine which document to retrieve.

Finally, we present the design and implementation of LARA, our own distributed crawler, which we used to develop our testbed.

Chapter 1

Introduction

The Web consists of large collections of hyperlinked resources, which web authors autonomously generate and modify. The Web contains a large volume of information on different topics, but, in contrast to traditional collections such as libraries, the Web has no centrally organised content structure. Furthermore, the autonomous nature of content creation leads to a high degree of volatility; documents are created, modified, and removed from the Web at a high rate. While it is these specific factors that have led to the Web's popularity, they also create significant challenges for people trying to locate information.

Locating resources that fulfil user information needs is not a trivial task, and search engines have become a critical tool for web users, with 3.6 billion searches conducted each month in the US alone [Comscore, 2004]. Many organisations deploy a similar information-seeking model within their own web sites, using a search engine within their intranets to provide searching capabilities to customers and staff.

The inverted file is the preferred data structure for efficiently processing queries submitted to search engines [Brin and Page, 1998; Witten et al., 1999]. An inverted file is similar to a book index (and is often simply referred to as "the index"), where, for each word, a list of the web documents containing that word is stored. To construct the index, a search engine must visit every document to be included in the index, parse the document, and add the relevant data to the inverted file.

It is the web crawler that collects documents from the Web and stores them locally in a centralised collection so that their content can be indexed and subsequently ranked in

response to user queries. These crawled documents are also used to generate document summaries — or snippets — that are returned as part of search results. Once they have been retrieved, crawlers must maintain these crawled documents by periodically revisiting the web documents they originate from. This ensures that the index and the summaries accurately reflect any changes that have been made to the documents since they were last crawled. Crawler accuracy or effectiveness is paramount to ensuring search user satisfaction: only resources that have been visited can be searched and resources should be up-to-date to meet users' information needs.

There are several approaches to predicting when a document is likely to change and therefore require recrawling. One method is to examine how frequently a document has changed in the past, that is, by examining the past change history. While there are many studies that have examined recrawl frequency when there is a long period of past change history available, this information is not always available, particularly when a document has only been crawled a few times. Little work has investigated schemes that work well when past change information is unavailable or limited.

Another approach for predicting when documents are likely to change is to use the `Last-Modified` and `Expires` HTTP headers that are returned along with web documents. When they are properly maintained and accurate, these headers can be a very effective method of improving the freshness of crawled documents. These headers, however, are not always available, and may be deliberately inaccurate to force web clients to update their locally cached copy of documents. In our work we examine different methods of predicting the likelihood that documents have changed and therefore require recrawling.

In this chapter we highlight some of the key aspects of web crawling, the impact that they have on maintaining the freshness of crawled documents, and how this can affect search engine user results. In the next section we discuss some of the key moments in the history of web crawling and show how crawling has evolved over time.

1.1 The History of Crawling

The first Internet "search engine", a tool called "Archie" — shortened from "Archives", was developed in 1990 and downloaded the directory listings from specified public anonymous FTP (File Transfer Protocol) sites into local files, roughly once a month [Deutsch,

1990; Emtage and Deutsch, 1991]. In 1991, "Gopher" was created, which indexed plain text documents. The programs "Veronica" and "Jughead" were used to search these Gopher indexes [Machovec, 1993; Jones, 1993; Harris, 1993; Hahn and Stout, 1994]. With the introduction of the World Wide Web in 1991 [Berners-Lee et al., 1992; Berners-Lee, 2008], many of these Gopher sites became web sites that were linked by HTML links. In 1993, the "World Wide Web Wanderer" was created, considered to be the first crawler [Gray, 1996b]. Although this crawler was initially used to measure the size of the Web, it was later used to retrieve URLs that were then stored in a database called Wandex, the first web search engine [Sonnenreich and Macinta, 1998]. Another early search engine, "Aliweb" (Archie-Like Indexing for the Web) [Koster, 1994a], allowed users to submit the URL of a manually constructed index of their site.

The index contained a list of URLs and a list of user written keywords and descriptions. The network overhead of crawlers initially caused much controversy, but this issue was resolved in 1994 with the introduction of the Robots Exclusion Standard [Koster, 1994b], which allowed web site administrators to block crawlers from retrieving part or all of their sites. Also, in 1994, "WebCrawler" was launched [Pinkerton, 1994], the first "full text" crawler and search engine. "WebCrawler" allowed users to search the content of web documents rather than the keywords and descriptors written by administrators, reducing the likelihood of misleading results and allowing greater search capabilities. Around this time, commercial search engines began to appear with Infoseek [2008], Lycos [2008], Altavista [2008], Excite [2008], Dogpile [2008], Inktomi [2008], Ask.com [2008], and Northern Light [2008] being launched from 1994 to 1997 [Sullivan, 2003b]. Also introduced in 1994 was Yahoo! [2008], a directory of web sites that was manually maintained, though later incorporating a search engine. During these early years Yahoo! and Altavista maintained the largest market share [Sullivan, 2003b]. In 1998 Google [2008] was launched, quickly capturing the market [Sullivan, 2003b]. Unlike many of the search engines at the time, Google had a simple uncluttered interface, unbiased search results that were reasonably relevant, and a lower number of spam results [Google, 1999]. These last two qualities were due to Google's use of the PageRank [Page et al., 1998] algorithm and the use of anchor term weighting [Brin and Page, 1998].

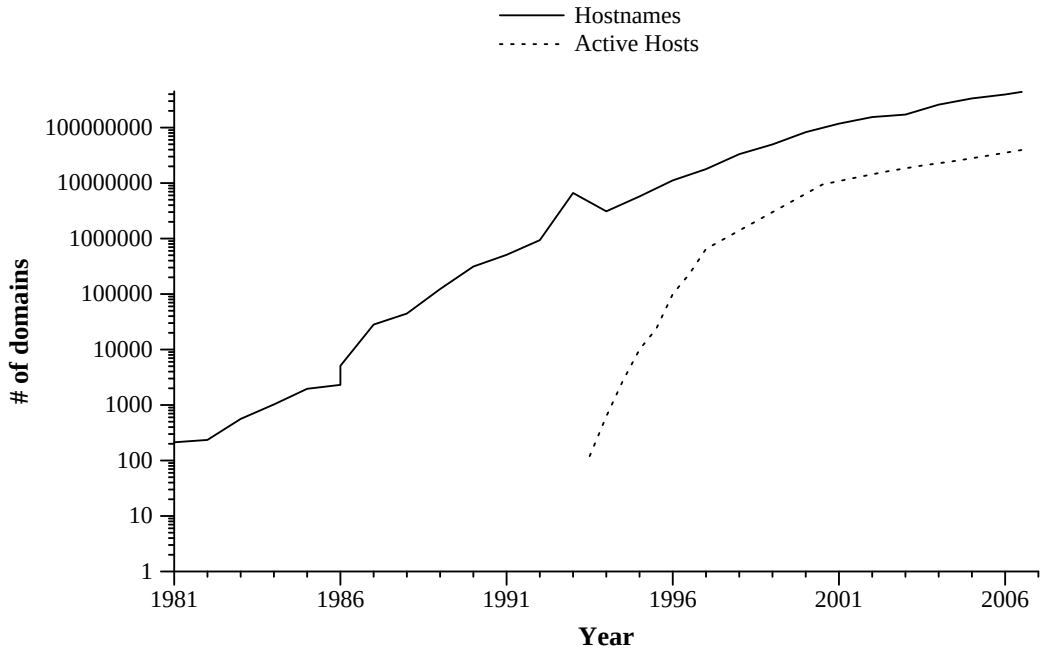


Figure 1.1: The growth in numbers of domain hosts and active hosts on the Internet from 1981 to 2006 [Gray, 1996a; ISC, 2006; Netcraft, 2006].

While early crawlers dealt with relatively small amounts of data, modern crawlers, such as the one used by Google, need to handle a substantially larger volume of data due to the dramatic increase in the size of the Web.

1.2 Size of the Web

Over the time frame of crawler development, the Web has been growing rapidly, and so crawlers need to operate efficiently and effectively [Brin and Page, 1998; Heydon and Najork, 1999; Shkapenyuk and Suel, 2002].

Generating statistics regarding the size of the Web is difficult since there is no single source of the Web's size over time. Statistics, need to be gathered from multiple sources, and these typically have differences in the way they are collected. Furthermore statistics regarding the number of web documents require complete crawls of the entire Web, which are both difficult and expensive to produce and maintain. An alternative indicator of the size of the Web is the number of domains or hosts on the Internet, however, not all domains have an associated public web site, and so, this is only an approximation of the number of web

sites. We present the estimated growth of domains and web sites in Figure 1.1. From this we can see that the number of domains follows an exponential growth, though this is beginning to slow down. Estimates of the size of the Web are varied though a recent study by Gulli and Signorini [2005] estimates that the indexable Web is more than 11.5 billion documents.

In the next section we describe the crawling process and highlight its interaction with the Web.

1.3 Web Crawling and the Web

The Web contains large volumes of documents and resources that are linked together. In the early days of the Web, manually locating relevant information was reasonably easy due to the limited amount of information that was available. Typically, users found relevant information with the aid of manually maintained web rings, links pages, and directories, such as Yahoo! [2008] and later DMOZ [2008], which were organised by topic. However, as the size of the Web grew, these approaches were augmented or replaced by automated systems using web crawlers and search engines.

Search engines typically support "bag of word" querying techniques, where users enter query terms and the search engine ranks web documents by their likelihood of relevance to the query terms. This approach, while effective, requires an index of documents on the Web. This index is created by retrieving a copy of every document to be indexed, from the Web, a task that is undertaken by a web crawler. Web crawlers exploit the link structure of web documents and traverse the Web by retrieving documents, extracting the embedded URLs, and following them to new documents. Retrieved documents are placed in a central repository so they can be indexed. Once indexed, a document is ranked in response to user queries and its URL is returned to searchers as part of a ranked list. The user then follows the link to the live Web copy of the document. We highlight this process in Figure 1.2 and the following example.

1. The crawler retrieves a document about a computer virus from the CNN home-page.
2. The crawler inserts the document into the local repository.
3. The search engine indexes the documents in the local repository.

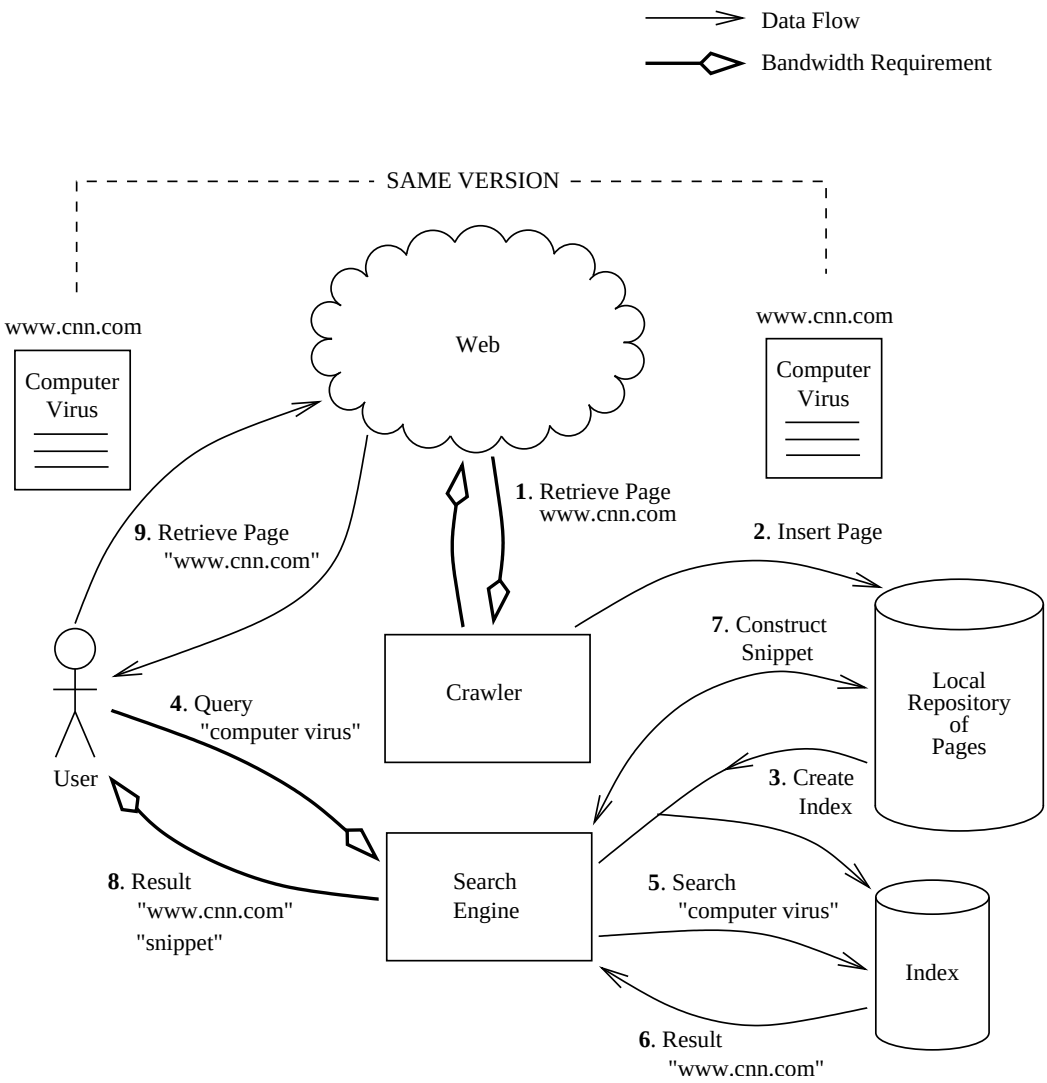


Figure 1.2: A typical querying process. The process begins with the crawler retrieving a document from the Web (1) and inserting it into a local repository (2). The document is then indexed (3), allowing users to search its contents. When users pose a query (4), the index is examined for relevant documents (5), which are ranked (6), have snippets extracted (7), and are returned to the user (8), who then follows the URL link to the live Web copy of the document (9).

4. A user poses the query "computer virus".
5. The search engine examines the index for relevant documents.
6. The search engine locates the CNN home-page and retrieves its URL.
7. The search engine creates a short snippet or summary from the cached document.
8. The search engine returns the CNN home-page URL and snippet to the user.
9. The user clicks on the URL and is presented with the CNN home-page containing the computer virus article.

However, the Web is a volatile environment where documents are frequently created, modified, and removed, which means that crawlers must revisit documents periodically to update the local repository. Index inconsistency occurs when crawlers fail to recrawl documents that have changed. In this situation the returned documents are no longer relevant to the user's query, a problem that we highlight in greater detail in the next section.

1.4 Web Crawler Index Inconsistency

Ideally the index of a search engine should represent a perfect snapshot of a web at the time each query is issued. In the case of the Web this is not feasible, as millions of queries are issued per hour [Sullivan, 2006b], and hundreds of millions of web documents are created every week [Ntoulas et al., 2004a]. To meet this demand, indexes would have to be updated at least several hundred times per second, which is not possible with current technology.

Given the dynamic nature of web data, a crawler must visit documents on the Web frequently in order to ensure that the index is up-to-date [Cho and García-Molina, 2000a]. In Figure 1.2, we presented a typical querying process. In Figure 1.3, instead we highlight the potential failure of the querying process with the following example.

1. The crawler retrieves a document about a computer virus from the CNN home-page.
2. The crawler inserts the document into the local repository.
3. The search engine indexes the documents in the local repository.

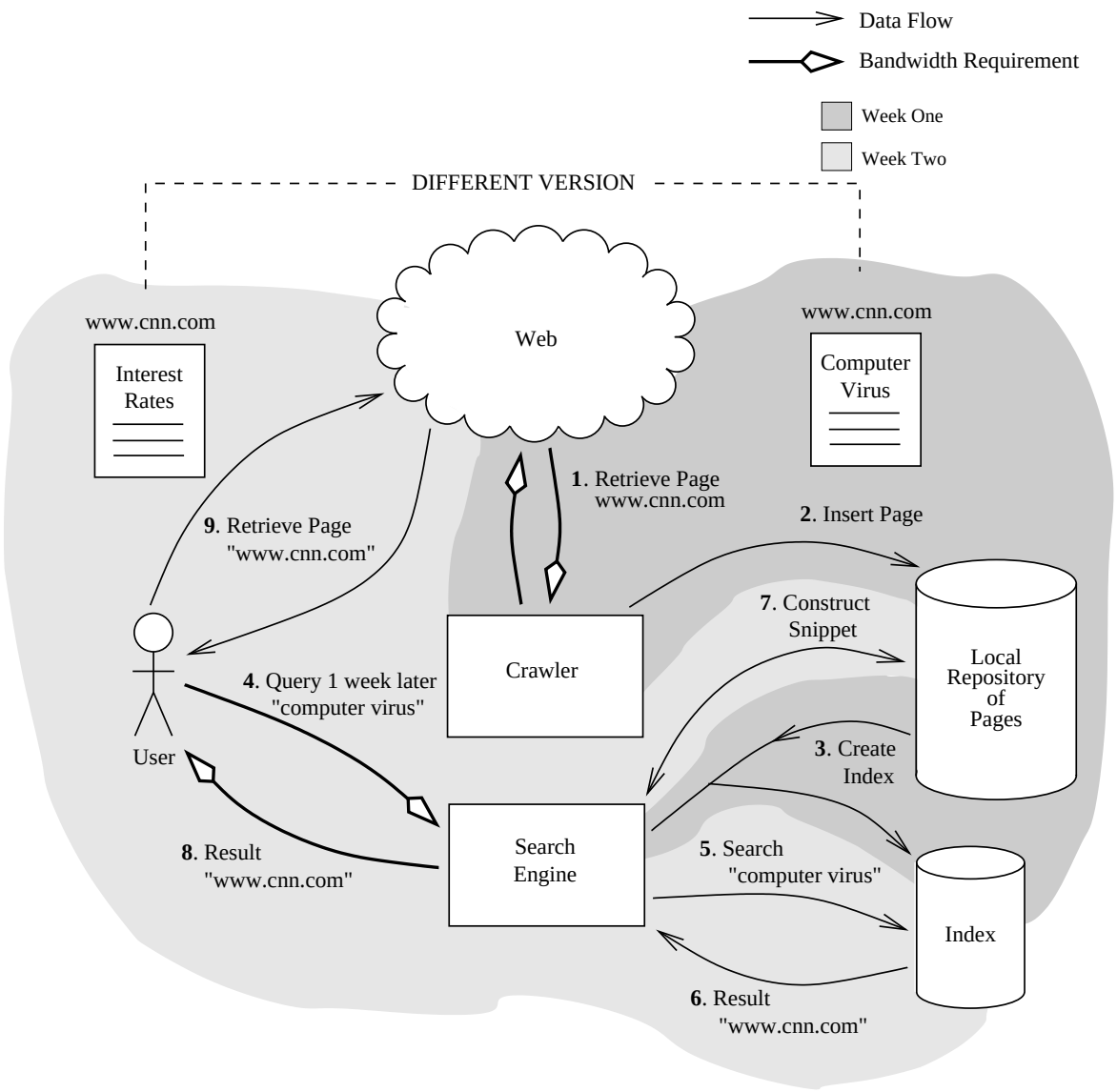


Figure 1.3: Failure of the querying process. Again, the process begins with the crawler retrieving a document from the Web (1) and inserting it into a local repository (2). The document is then indexed (3), allowing users to search its contents. However, this time the user poses the query one week later (4). As before, the index is examined for relevant documents (5), which are ranked (6), have snippets extracted (7), and are returned to the user (8), who then follows the URL link to the live Web copy of the document (9). However, in the week since the document was last crawled, the live Web copy has changed, and is not relevant to the query.

4. A week passes by before a user poses the query “computer virus”.
5. The search engine examines the index for relevant documents.
6. The search engine locates the CNN home-page and retrieves its URL.
7. The search engine creates a short snippet or summary from the cached document.
8. The search engine returns the CNN home-page URL to the user.
9. The user clicks on the URL and is presented with the CNN home-page, however, the document has changed, and now discusses “interest rates”.

In our original example, an online news document discussing a computer virus is retrieved by a crawler and is indexed by the search engine. This article was correctly returned as a relevant match to a query about “computer viruses”. However, in our second example, a week has gone by since the document was crawled and now the online article has changed, instead discussing “interest rates”. Since the crawler has not retrieved an updated copy of the document, any queries regarding “computer viruses” will still match the indexed document. When users retrieve the online document, however, it is likely to be irrelevant to their information need. Furthermore, any queries about “interest rates” will not return this document, since the index does not contain information from the more recent article.

A search engine, therefore, must divide its bandwidth between two tasks: answering users’ queries and crawling the Web to find and update locally stored information. In order to provide fast answers to users’ queries, as little bandwidth as possible should be spent on crawling, rather devoting the resources to query processing. However, if the crawler does not revisit sufficient documents in the Web, the index may not contain new documents that have appeared in the Web since the last crawl, and may contain old information about existing documents that have been edited since last crawled.

While revisiting documents infrequently produces index inconsistency, a crawler should not revisit documents that have not changed; this wastes the limited bandwidth that the search engine must divide between processing queries and crawling resources. A naïve crawler that revisits documents randomly is likely to waste the search engine’s limited bandwidth retrieving documents that have not changed. Therefore, a crawler needs to be able to predict which documents will change, and revisit only those documents. Ideally a crawler should

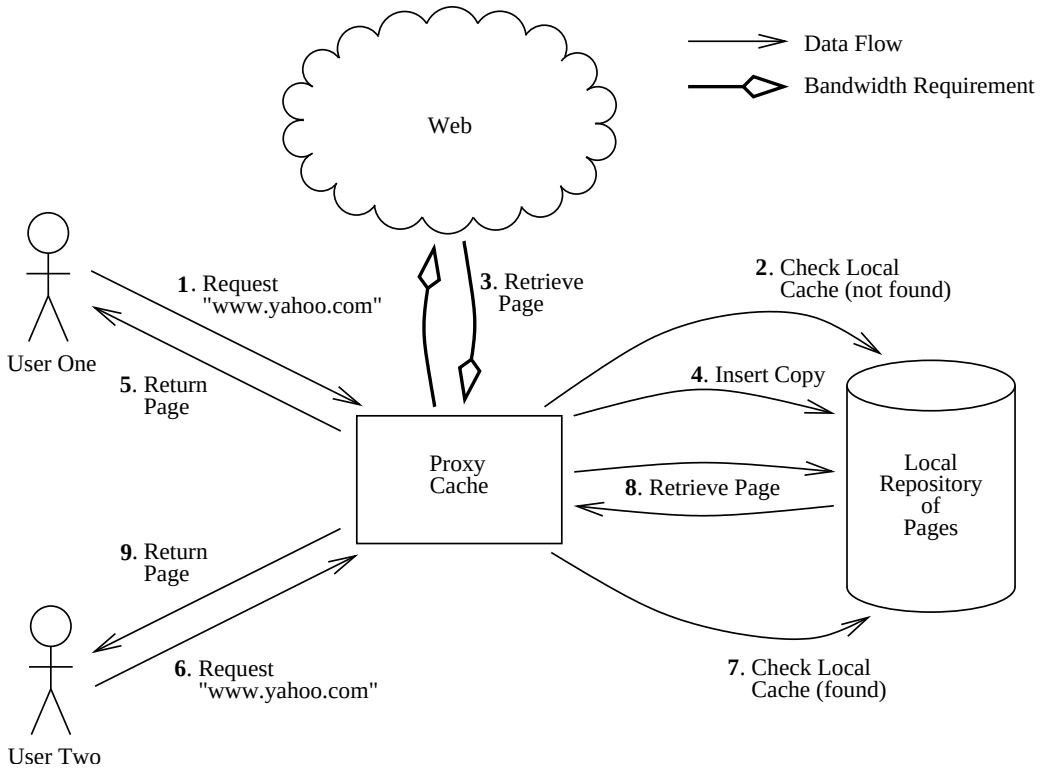


Figure 1.4: A typical proxy cache process. A user makes the initial request for a document from the web proxy (1). The proxy checks the local repository for the resource, but does not find it (2). The proxy requests the resource from the Web (3), puts a copy into the local repository (4), and returns the resource to the user (5). Later, another user requests the same resource from the proxy (6). When the proxy checks the local repository, it finds a copy (7), which it retrieves (8) and returns to the user (9). While the initial request for the resource results in the proxy retrieving a copy from the Web. Subsequent requests are fulfilled with the locally cached copy, resulting in reduced web traffic.

only revisit documents that have changed and that will contribute to search results of some future queries. We explore this stricter requirement on document change in Section 5.4.

A similar problem to search index inconsistency is studied in the proxy cache literature and deals with the issue of maintaining proxy cache consistency and coherency [Srinivasan et al., 1998; Belloum and Hertzberger, 2002]. However, unlike proxy caches, crawlers generally do not consider cache replacement issues [Podlipnig and Böszörményi, 2003].

1.5 Proxy Caching

Proxy caching is a technique that reduces network traffic and its associated cost by maintaining a local copy of popular resources.

We highlight a typical proxy caching process with the example in Figure 1.4. In this example there are two different users on the local network.

1. The first user requests the Yahoo! home-page from the web proxy.
2. The proxy first checks the local repository for the resource, but does not find it.
3. The proxy then requests the resource from the Web.
4. After retrieving a copy of the resource, the proxy places a copy into the local repository.
5. The proxy then returns the resource to the user.
6. Later, a second user requests the same resource from the proxy.
7. The proxy checks the local repository, this time finding a copy.
8. The proxy retrieves a copy from the repository.
9. The proxy then returns the resource to the user.

From this example we can see that while the first request for a resource required that the proxy retrieve the resource from the Web, subsequent requests could be satisfied with the locally cached copy, saving traffic costs.

While proxy caching reduces network traffic, it can suffer from the same inconsistency problems as search indexes. Since documents are cached locally, there can be differences in the local copy and the source on the Web. We highlight this with the example in Figure 1.5.

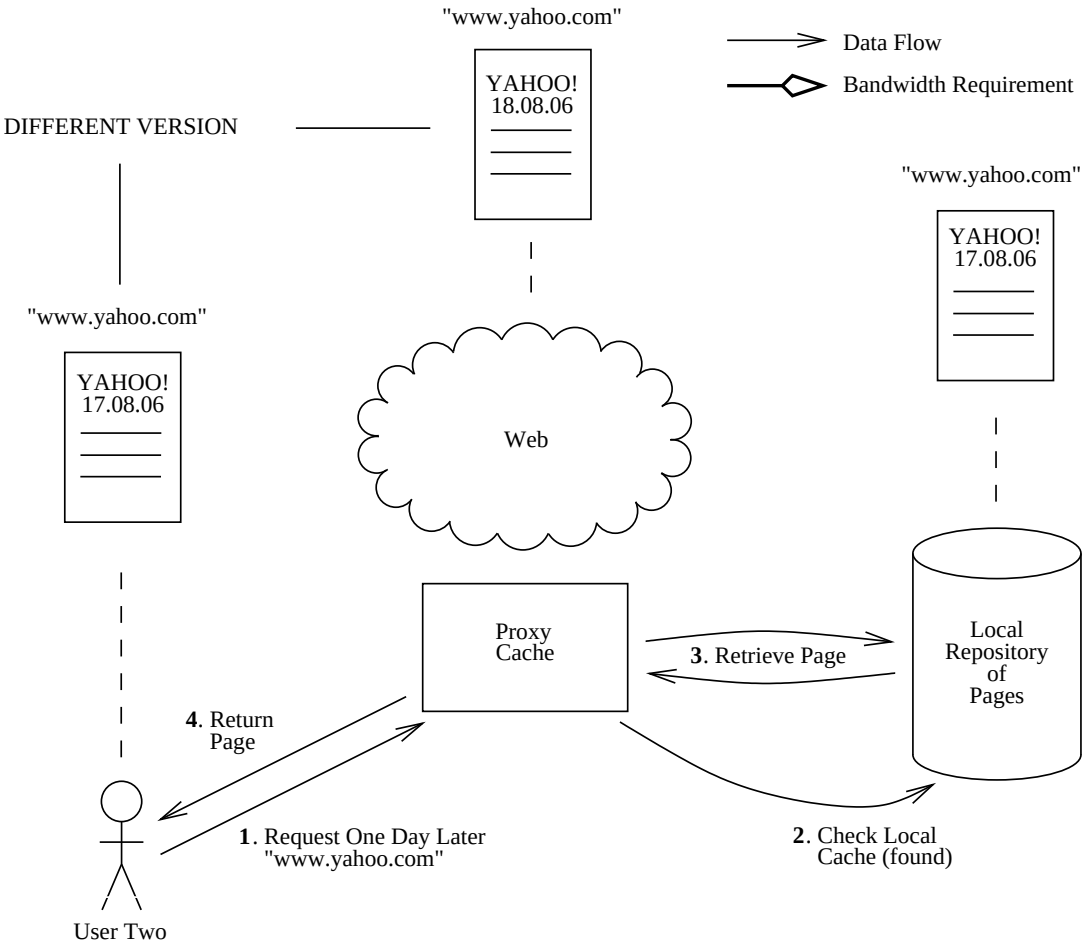


Figure 1.5: Proxy cache failure. One day later, another user requests the same resource from the proxy (1). When the proxy checks the local repository, it finds a copy (2), which it retrieves (3) and returns to the user (4). Hence a request by a user for a resource that was first requested on the previous day is fulfilled by the local cache. Unfortunately, the source on the Web has changed and no longer matches the cached version. As such, the user is returned a stale (out-of-date) version of the resource.

Again, we have a user on the local network requesting the same resource from the previous example in Figure 1.4, however, this time the request is one day after the original request.

1. A user requests the Yahoo! home-page.
2. The proxy checks the local repository, again finding a copy.
3. The proxy retrieves a copy from the repository.
4. The proxy then returns the resource to the user.

However, as we can see in the figure, the original source on the Web has been modified. Meanwhile, the locally cached copied has not been updated, and hence, the user is returned a stale (out-of-date) copy of the resource. To avoid this problem, the proxy needs to be able to determine when locally cached copies of resources are likely to become stale. Proxy caches typically use the `Last-Modified` and `Expires` HTTP headers to determine when this is likely to occur.

While proxies operate in a passive role, acting in response to user requests for documents, web crawlers operate in a more active way. Crawlers do not wait for user requests, instead, using heuristics to determine which documents to crawl. A common approach among many crawlers is to examine the URL link structure of web documents and decide which document to crawl based on link characteristics.

1.6 Web Structure

The Web can be thought of as a graph, where documents are nodes (or vertices) and hyperlinks are edges. This graph structure has been examined in many studies to try and discover insights into algorithms for crawling and searching.

A study by Broder et al. [2000a;b] examines the properties of two Altavista crawls in 1999, with over 200 million documents and 1.5 billion links. They confirm that the power law for in-degree holds true for the Web in general [Kumar et al., 1999; Barabási and Albert, 1999; Huberman and Adamic, 1999]. They also show that over 90% of the Web falls into four main components, as shown in Figure 1.6. The first piece is the *central core* which consists of documents that are strongly connected. That is, given any two documents (u and v) there is

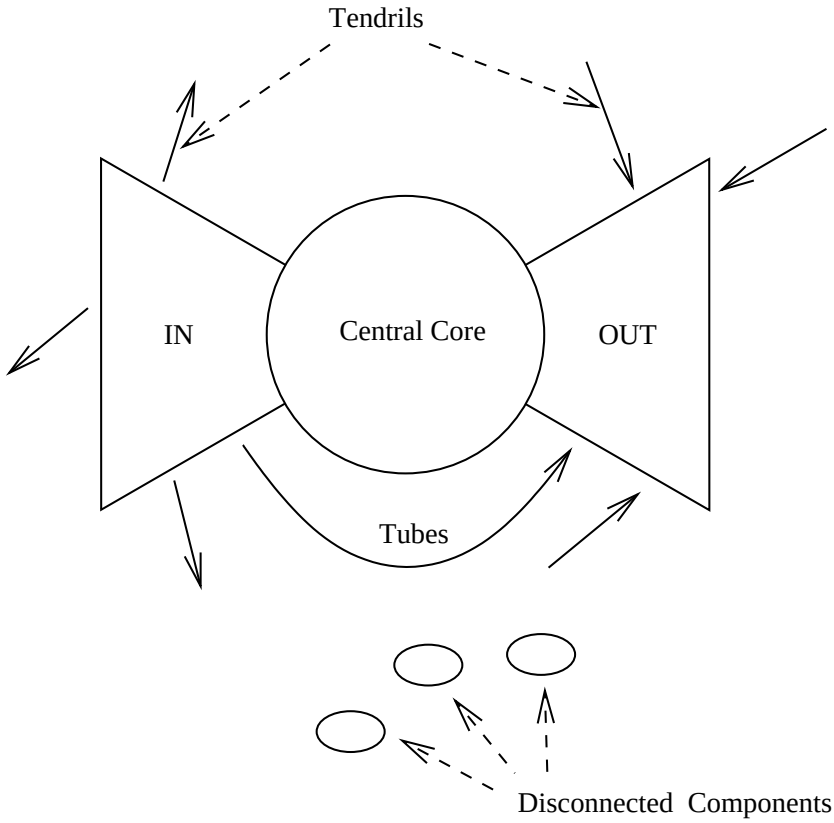


Figure 1.6: The connectivity and macroscopic structure of the Web. The Web is broken down into four main components, the central core, the IN group, the OUT group, and tendrils. The central core is highly connected. The IN group links to the core but is not linked to by the core. The OUT group is linked to by the core but does not link to the core. Tendrils are document that belong to the IN or OUT group but do not link to the central core. Two additional types are tubes and disconnect component. Tubes are documents in the IN group that connect to the out group but not to the central core. The disconnect component is the set of documents that are not linked in any way to connected component of the Web.

a path from u to v and from v to u. The path, however, may not be direct and instead may be via intermediate documents. The second and third group of documents belong to the *IN* and *OUT* group. The *IN* group consists of documents that link to the *central core* but are not linked to by the core. These could be new, unknown or unpopular documents. The *OUT* group consists of documents that are linked to by the *central core*, but do not link back to the core, and are most likely corporate documents that contain internal links only. Finally, there is a *tendrils* group that consists of documents that do not link to, and are not linked by, the *central core* group of documents. Documents in the *tendrils* group hanging off the *IN* group may link to the *OUT* group directly without connecting to the *central core*. These documents are classed as *tubes*. The remaining documents consist of *disconnected components* that are not linked to or linked by any documents in the connected component of the Web.

A document that belongs to the *IN* group can be used as a seed URL with which to begin the crawling process, and successfully reach most documents in the connected component of the Web. To guarantee that all documents were retrieved, every document in the *IN* group would need to be used as a seed to begin the crawl. If the crawl begins from any other group of documents, a smaller subset of documents in the connected group will be visited.

In contrast, Diligenti et al. [2004] examine random walks on a web graph and compare various document ranking schemes on both general crawls and topic specific crawls. As we discuss in Section 2.11, topic-driven crawling attempts to retrieve documents that are about a specific theme. This is distinct from general crawling schemes that attempt to crawl all documents regardless of their topic. Diligenti et al. propose a framework that incorporates many of the published ranking schemes, including PageRank [Page et al., 1998], as well as Hubs and Authorities [Kleinberg, 1999]. In the next section we give a brief outline of this thesis and show how our work fits in with previous work in crawling.

1.7 Overview of Thesis

The aim of the crawling process is to retrieve as many of the documents on the Web (and enterprise webs) that will be returned in response to user queries and that users will find relevant to their queries, while keeping the collection current and minimising the resources used in the process. Crawler efficiency is important: network bandwidth is expensive; careful

rules are needed to avoid traps where a crawler continually revisits the same resource; and resources should only be revisited when there is a likelihood that they have changed.

A web crawler initially retrieves resources that are manually selected and submitted by users (who may be internal maintainers of the search engine or external authors of web resources), processes those resources and extracts hypertext links, and continues the process, retrieving resources as their URLs (Uniform Resource Locators) are extracted.

The crawling process is outlined in Figure 1.7.

1. The crawler removes the highest-ranked URL from the *frontier*, a list of unvisited URLs.
2. The document is retrieved from the Web.
3. A copy of the document is placed in the local repository for indexing by the search engine.
4. The crawler parses the document and extracts the HTML links, with each extracted URL converted to a standardised format.
5. The extracted URLs are compared to a list of all previously extracted URLs ("All URLs" list in Figure 1.7) and any new URLs are added to this list.
6. If the URL is added to the list of all previously extracted URLs, it is also inserted into the *frontier* for crawling.
7. The *frontier* is reordered using some scheme [Cho et al., 1998], such as breadth-first ordering or PageRank [Page et al., 1998], and the process repeats from Step 3 until the *frontier* is empty, the crawl repository is full, or the resources allocated to crawling are exhausted.

In this thesis, we consider part of the web crawling problem. After crawling to establish a collection, resources need to be recrawled periodically so that the stored copy is refreshed to match the actual resource on the Web.

Specifically, we investigate:

- Past work on improving crawler effectiveness and efficiency, both in terms of collection quality and freshness through crawl ordering.

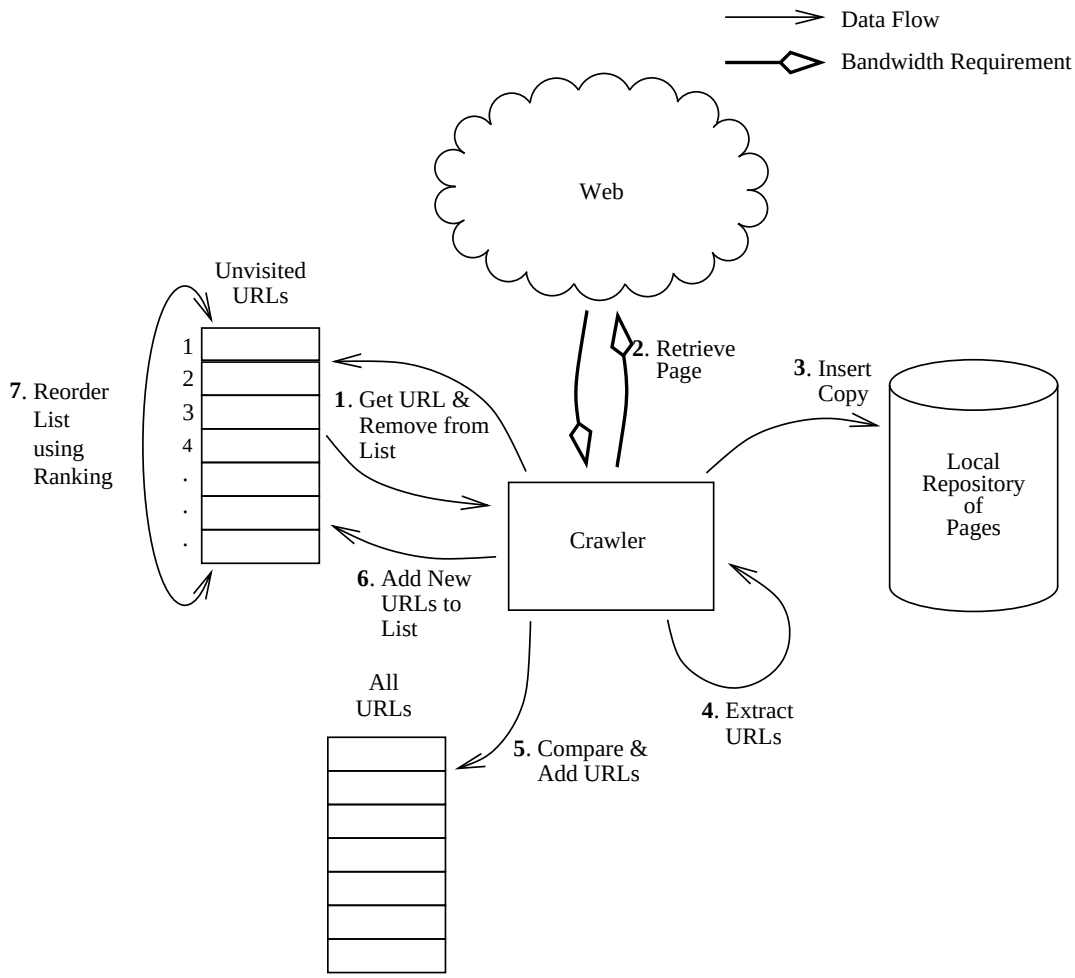


Figure 1.7: The crawling process. The unvisited list initially contains seed URLs that have been manually chosen. The crawler removes the top ranked unvisited URL (1), retrieves a copy of the document from the Web (2), and inserts the copy into the local repository (3). The crawler then parses the HTML document and extracts the URLs it contains (4). Each extracted URL is compared to a list of previously extracted URLs (5) and any previously unseen URLs are added to both the all URL list and the unvisited URL list (6). Finally, the crawler reorders the unvisited URL list (7), and the process repeats, until either there are no unvisited URLs left, the repository is full, or a preset number of documents have been crawled.

- A framework that allows for consistent evaluation of crawler performance.
- A metric for detecting and predicting the impact of change on user search results.
- The effectiveness of various stateless crawl techniques with regard to collection freshness.
- Adaptive crawl ordering schemes and their impact on collection freshness.
- Issues relating to the design and development of a crawler.

While various approaches have been proposed that effectively maintain collection freshness with a long history of change, our work focuses on schemes that do not rely on a history of change. One possible approach to establishing when to recrawl resources is to manually or automatically classify documents based on their content or URL; for example, a list of well-known news sites might be manually maintained, and all resources within the list recrawled every 12 hours. Another approach is to retrieve a resource twice, to determine how much the resource has changed, and use this information to establish a recrawl frequency. We focus on elements of the latter approach in Chapter 4 but, in practice, both approaches may be applied and we investigate these various approaches in Chapters 5 and 6.

In this chapter we introduced the Web, the crawling process, and provided a brief overview of the problems related to maintaining consistency of crawled collections. In Chapter 2 we provide more detail on the difficulties related to crawling and describe the various work that has been conducted on the topic of crawling.

We investigate previous work that has measured the effectiveness of a crawler in different ways, such as, a crawlers ability to maintain index *freshness* [Cho and García-Molina, 2000a; Edwards et al., 2001], that is, return documents that have changed since the last crawl [Pandey and Olston, 2005; Ghodsi et al., 2005]; a crawlers ability to return *important* documents [Cho et al., 1998; Najork and Wiener, 2001], typically defining a document as important if it will appear high in a ranked list that uses the PageRank algorithm [Page et al., 1998]; a crawlers ability to return documents about a particular topic, through ordering of the crawl by its relevance to a topic, otherwise known as *topic-driven crawling* [Chakrabarti et al., 1999]; a crawlers ability to reduce search engine *embarrassment* [Wolf et al., 2002]; a crawlers ability to crawl the *hidden web* [Raghavan and García-Molina, 2001], which attempts

to retrieve content "hidden" behind search forms in searchable databases; and methods for implementing *parallel crawling* [Cho and García-Molina, 2002], which investigates ways of efficiently running multiple crawlers. To determine the ranking, typically, these evaluation schemes ignore whether the document returned by the crawler will ever be read by a user.

Throughout this thesis, we examine ways of ordering and evaluating a crawl to increase the likelihood of retrieving new and changed documents that are likely to be useful to search engine users. Our evaluation methods in Chapter 4, for instance, incorporate query ranking to determine the effect that change metrics have on user results, while our methods in Chapter 5 examine the *popularity* of documents with users and incorporate this information into the crawl evaluation.

In Chapter 3 we introduce our experimental environment, highlight our need for a test collection and how we gathered this collection from the Web.

In Chapter 4 we discuss our document change metrics, comparing them to other change measures and investigating their effectiveness in measuring changes that affect search engine results. We demonstrate that they are more effective than the commonly used metrics. We also present our evaluation techniques.

Then in Chapter 5 we examine various "stateless" crawl ordering schemes that are widely used, primarily to improve collection quality, and we study their impact on index freshness.

We incorporate popularity to ensure that the utility of retrieved documents is considered as part of the performance evaluation. We consider the popularity of a document by examining the number of times the document has been returned as an answer to some past user queries of varying scope. Obviously a crawler is wasting bandwidth if it finds important, new documents but these documents are never returned to any user. Using the metrics of freshness and popularity, we examine algorithms employed by crawlers to find documents with minimal waste of bandwidth. We use algorithms that have been proposed in the literature, and introduce several of our own. We show that no one scheme performs best on all collections, but that query statistics are an important factor when maintaining the effectiveness of search results.

In Chapter 6 we investigate our novel approach to crawl ordering, which uses anchor term statistics, in conjunction with stateless crawl ordering schemes, to bias a crawl towards changed, new, and important documents. We find that it can improve collection freshness,

while maintaining the properties of different static schemes, particularly when combined with our dynamic scheme adaptation approach.

One of the important functions for a crawler is to maintain the URL *frontier* — a list of URLs parsed from retrieved documents that the crawler has not yet visited [Chakrabarti, 2002]. Several previous studies have examined methods for biasing the frontier towards changed and new documents, but unlike our scheme in Chapter 6, have not used anchor text as a basis for the bias. One line of investigation has been the derivation of mathematical models of change, allowing an optimum crawling strategy to be derived for that model. For example, past studies have developed optimal strategies for crawling documents that change according to a Poisson process [Cho and García-Molina, 2000a], or a quasi-deterministic model [Wolf et al., 2002]. While studies have shown that documents can change according to a Poisson model [Brewington and Cybenko, 2000b], other larger studies have shown that most documents are modified during US working hours [Brewington and Cybenko, 2000a], and hence, change on the Web is not a Poisson process. There have been other studies that have produced models without these assumptions that have performed well, albeit against simulated data [Edwards et al., 2001]. In our work we specifically focus on crawling collections where there is insufficient data to fit an accurate mathematical model of change, and where recrawling static documents is extremely undesirable.

Several schemes have been used in the past to determine when documents are stale and hence need to be updated. The simplest and most widespread method uses the `Expires` and `Last-Modified` HTTP headers. If the current date is after the `Expires` date or the difference between the current date and the `Last-Modified` date exceeds a threshold, the copy of a document is considered *stale* (does not match the original) and needs to be updated. Proxy servers such as Squid [Wessels, 2001b;a] examine the `Last-Modified` and `Expires` HTTP headers to determine when to update their local copy of a document. Unfortunately, the `Last-Modified` and `Expires` HTTP headers can be inaccurate or missing [Wills and Mikhailov, 1999a]. Some web administrators deliberately supply misleading headers to encourage web clients to update their local copy of a document.

An alternative method for improving collection freshness is to use past change to predict future change [Fetterly et al., 2003b; Ali and Williams, 2003; Ntoulas et al., 2004a]. That is, if there are two copies of each document, from two separate crawls, a third subsequent crawl

can be ordered on the basis of whether each document has changed or not. This is based on the assumption that documents that have changed in the past are more likely to change again. While this method has been shown to perform well, it has one major drawback. It requires two complete crawls of all documents before any document change frequency statistics are available, and so, only on the third crawl can the frontier be reordered to improve collection freshness and crawler efficiency. Furthermore, this scheme can only adapt in response to past change statistics. While it will work well if past change statistics reflect the current change statistics of each document, they must remain stable. It cannot adapt dynamically to documents that change at irregular intervals. Our schemes in Chapters 5 and 6, on the other hand, do not rely on past change to predict future change.

Finally, in Appendix A, we present the design issues that were considered as part of our development of the Lara crawler, which we used to gather our test collections.

Chapter 2

Background

There are many elements to crawling, as introduced in the previous chapter. Here we consider crawling in detail, considering HTTP headers and HTML markup, web crawling, observing document change, and how the Web changes. We conclude with a survey of past work on crawling.

2.1 Data on the Web

The crawlers described in the previous chapter, such as Archie, indexed text only, however these days search engines index many different document formats, such as HyperText Markup Language (HTML), Adobe Portable Document Format (PDF), and PostScript (PS) files. Here we provide a brief outline of the structure of HTML web documents and how they relate to web crawlers. We only cover the structure of HTML documents since they contain an extensive link structure, make up a large proportion of crawled documents, are well studied, and are very different to other document formats in terms of dynamics [Douglis et al., 1997; Wills and Mikhailov, 1999b; Zander et al., 2003].

2.1.1 HTTP Headers and HTML Markup

Web resources that are indexed by search engines typically contain two components: HTTP headers and the resource content. HTTP headers specify information that includes the date the document was returned, the web server, the HTML document length, and the document content type [Fielding et al., 1999a]. The resource content is often marked-up

using HTML [W3C, 1999], although it may also be another content type such as Adobe PDF format [Corporate Adobe Systems Inc., 1993], PostScript [Corporate Adobe Systems Inc., 1999], or a proprietary text format. When HTML is used, the markup specifies the HTML recommendation, the document character set (which could alternatively have been specified in the HTTP headers), and the structure of the document [Fielding et al., 1999a]. We focus on HTML resources in our work.

Web crawlers make particularly use of the `Last-Modified`, `Expires`, and `Date` HTTP headers. The `Expires` header explicitly indicates the date at which the document should be recrawled. When used correctly, this header solves the recrawl problem, that is, it indicates a validity period for the document version and an exact date for recrawling. However, as we show experimentally later in Section 4.1, it is an unreliable tool: many web servers do not provide the header and, when it is provided, it is often set to a past time. Setting the `Expires` header to a past time helps ensure a document is not cached by user agents such as web browsers and proxy caches [Fielding et al., 1999b].

The `Last-Modified` and `Date` headers are often provided. As we discuss in Section 4.3, if two versions of a document are available, then past document changes can be used as a predictor of a future change. In this context, the time interval between the current retrieval — specified in the `Date` header — and the most-recent change to the document — specified by the `Last-Modified` header — can be used as inputs to a predictive function [Brewington and Cybenko, 2000b] to decide when the document is likely to change. We would expect that this approach would be effective when the time between retrieval of document versions is short.

The use of the `Last-Modified` header to predict document freshness is an approach used extensively by proxy caching systems [W3C, 1995; Wessels, 2001a] such as Squid [Wessels, 2001b]. This approach, however, relies on the accuracy of the HTTP headers. Previous studies have examined the accuracy of headers [Mogul, 1999] and their implications for caching [Wills and Mikhailov, 1999a], finding that:

- 38% of HTTP responses had HTTP date headers that are impossible.
- 3% of documents with no change in the Last-Modified field had changed.
- 36% of documents had changed, despite no change in the Last-Modified field.

```
<HTML>
  <BODY>
    <H3>
      A sample HTML document with a link to the
      <A HREF="http://www.rmit.edu.au/">RMIT Home-Page.</A>
    </H3>
  </BODY>
</HTML>
```

Figure 2.1: A simple example of HTML.

- 41% of documents with an expires and date field were "pre-expired", that is, they had an expiry date in the past.

In another study, Wills and Mikhailov [1999a] find that more than 9% of resources did not change despite a change in the `Last-Modified` header, and 0.3% of resources had changed when the header had not. They also find that in 14%–18% of cases no `Last-Modified` header was available.

In addition to studying change and HTTP headers, Wills and Mikhailov investigate actual document change. They find that many document changes were predictable, that is, the same lines in the HTML markup often changed as, for example, banner advertisements were rotated. This observation is consistent with our motivation and conclusions that document content needs to be analysed to determine the importance of change in web documents. We investigate this further in Section 4. Also of particular importance to crawlers is the anchor text contained within the HTML documents, which we discuss next.

2.1.2 Anchor Text

HTML resources, as exemplified in Figure 2.1, are typified by markup contained within angle brackets "< >", known as tags. While HTML contains large amounts of markup, that is, tags that define how content should be presented, much of this was not actually useful to crawlers or search engines until only recently. Search engines typically only index the textual content that will be visible on a web browser and ignore other content [Sullivan, 2002].

One major exception is that techniques of "spamming" search engines have forced search engine engineers to pay closer attention to formatting to avoid documents that are misleading and attempt to promote their rank. For instance, modern search engines consider how prominent the text is on a web document when evaluating its significance, for instance, ignoring text that has the same colour as the background or that is in a small unreadable font [SEO Logic, 2006].

Of significant importance to both crawlers and search engines are HTML anchors and anchor text. Anchors are the link information that connects web resources to each other. These links allow users to traverse from one HTML resource to another. Crawlers also follow these same links to traverse web documents so that they can be indexed by search engines and subsequently searched by users.

Anchor text is the text that appears highlighted as a link to other documents or resources, and generally provides information about the linked resource. For example, in Figure 2.1, we can guess that the URL belongs to the RMIT home-page without visiting the site, because the anchor text suggests this. Of course, we would not trust this assumption if we only had one document to go by, but, the large volumes of anchor text that are collected during a crawl can be correlated with each other, providing a good indication of document content [Amitay, 1998; Craswell et al., 2001; Upstill et al., 2003b]. They represent a third-party judgement of the content of a document and a summary of its content. Many search engines rank their importance highly and use them to judge the importance of web documents [Brin and Page, 1998; Sullivan, 2006a]. When search engines do not have the resources to visit all documents, they may resort to indexing only the anchor text that has accumulated for a document, allowing users to search a document that the crawler has not retrieved [Brin and Page, 1998; Hawking and Thomas, 2005]. Furthermore, *topic-driven* or *focused crawling* depends on anchor text to order the crawl to find documents that are about a given topic [Chakrabarti et al., 1999; Diligenti et al., 2000; Chau and Chen, 2003]. We discuss topic-driven crawling in more detail in Section 2.11. While both anchor text and textual content are both indexed by search engines, several preprocessing techniques are applied to improve their utility. For instance, when crawlers need to correlate large volumes of anchor text phrases to determine their frequency, preprocessing techniques help match related anchors. We discuss these techniques in more detail in the next section.

2.1.3 Text Preprocessing

Many information retrieval (IR) schemes use text preprocessing and normalisation techniques to improve their effectiveness [Baeza-Yates, 2004]. IR techniques rely on term statistics to predict the relevance of query terms to documents in a collection.

However, for term statistics to be generated, term delimiters must be defined [Williams and Zobel, 2005]. While this is relatively straightforward for English documents, as words are separated by well-defined punctuation, this is much more difficult for other languages such as Chinese [Chen and Liu, 1992; Zhang et al., 2005]. Once terms are extracted, they must be preprocessed to ensure that semantically similar terms are associated accordingly. Stemming, stopping, and case-folding are used by search engines to increase the likelihood of matching related terms [Witten et al., 1999; Baeza-Yates and Ribeiro-Neto, 1999].

Case-folding involves converting all characters to the same case, either uppercase or lowercase. This ensures that terms will match regardless of their case.

Stopping is a process that removes common or function words that usually add no new information to a sentence [Witten et al., 1999], and are typically conjunctions, such as *AND*, *OR*, and *THE*. Stop words occur in nearly all documents in the collection, and so, cannot be used to distinguish different documents. Removing stop words reduces computational load and typically has very little or no impact on effectiveness [Baeza-Yates and Ribeiro-Neto, 1999; Lester, 2006] . When comparing anchor phrases, stopping would be expected to improve the correlation of related anchors, since anchor phrases would be more likely to match. For instance, the following three anchor phrases will match after stopping:

```
"~~To~~ ~~the~~ RMIT home-page"
"~~This~~ ~~is~~ ~~the~~ RMIT home-page"
"RMIT home-page"
```

Stemming reduces terms to their stem or root word by removing a suffix, so that words that are semantically similar are reduced to the same word [Frakes, 1992]. The resulting stem is not necessarily a real word, for instance, the words *computing*, *computer*, *computers*, and *computation* are all reduced to the stem *comput* through the removal of the relevant suffix: *-ing*, *-er*, *-ers*, and *-ation*. The stemming process needs to be repeatable and preferably the

resulting stem is unique for each semantic concept. In Chapter 6, which describes work based on anchor text, we stem with an implementation of the Porter 2 stemmer [Porter, 1997].

Once terms are stemmed, stopped, and case folded, their frequency and distribution can be analysed further to determine the likelihood that documents are relevant to user queries and rank them accordingly using a document ranking function. Document ranking functions are important since crawling algorithms can use feedback from query results to help improve the crawling process, and in return produce better query results [Wolf et al., 2002; Bullot et al., 2003; Pandey and Olston, 2005]. Furthermore, we use document ranking functions throughout our work. In Chapter 4, we use document ranking functions to determine the effect of change metrics on query results, and in Chapter 5 we use document ranking functions to determine document importance.

We discuss crawling and querying processes further in the next section and highlight the impact they have on each other.

2.2 The Crawling and Querying Process

As discussed in Section 2.1.2, the link structure of web documents allows users to traverse between HTML documents. While users have a web browser to render the HTML document and they traverse the documents by clicking on the anchor links, crawlers must parse the HTML document and extract the HTML links contained within anchor tags. Furthermore, each crawled document must be indexed to allow users to search them.

A crawler's basic function is to request HTML web documents from sites on the Web, extract URLs from these documents, and maintain a *frontier*.

The steps in the crawling process are:

1. The crawler *frontier* is initialised with a set of seed URLs with which to begin crawling. This may be provided by the crawler's administrator or may be submitted by web site administrators.

2. The *frontier* is reordered according to some scheme [Cho et al., 1998], ranging in complexity from simple breadth-first ordering, to more complex schemes such as PageRank [Page et al., 1998]. When applying schemes, such as PageRank, that use the link

structure of documents, a partial web graph is created from the documents that have been crawled up to that point.

3. The highest-ranked URL is visited and removed from the *frontier*.

4. A copy of the retrieved resource is inserted into the central repository for indexing by the search engine.

5. The document is parsed and HTML links in the resource are extracted.

6. Each extracted URL is converted to a standardised format and compared to a list of previously extracted URLs.

7. If the URL has not been previously extracted, it is inserted into both the list of previously extracted URLs and the *frontier*, otherwise it is ignored.

8. The *frontier* is reordering and the process repeats from Step 3 until the *frontier* is empty, the crawl repository is full, or the resources allocated to crawling are exhausted.

The crawling and querying process are closely tied together. If a crawler performs poorly, the search results returned to users will be poor, as explained in Chapter 1. The querying process can provide key information about what documents and topics users find important over time [Google Zeitgeist, 2008].

The querying process begins once the crawler inserts a copy retrieved documents into the local repository. These documents must be indexed to allow their content to be searched. An index, much like the one found in a book, allows users to quickly locate documents that are relevant to particular terms, without resorting to scanning every document [Salton, 1971; Witten et al., 1999; Zobel and Moffat, 2006]. The first step is to normalise the content, as outlined in Section 2.1.3. Once this has been done, an *inverted index* is constructed. An inverted index contains a list of pointers to each occurrence of a term in a collection [Brooks, 2003]. Whenever users submit queries, the inverted index is used to locate relevant documents, which are ranked according to their likelihood of relevance [Salton and Buckley, 1988], which we discuss next.

2.2.1 Ranking Schemes

In the previous section we discussed the querying process, which returns documents in response to queries based on their likelihood of relevance. However, search engines require automated methods for ranking these documents, due to the overwhelming size of the Web and the large number of documents that need to be considered when ranking them in response to queries. In this section we outline some of the ranking methods that have been proposed in the literature. We break these into three types — those that are document ranking functions, those that are query independent, and those that are query dependent.

Document Ranking Functions

Document ranking functions determine the likelihood of a document being relevant to a query. They are the primary method for ranking query results in search engines and can use different techniques such as query similarity, probabilistic models, or language models to rank documents.

- **Term Frequency – Inverse Document Frequency**

  Term Frequency – Inverse Document Frequency (*tf.idf*) weights terms so that distinguishing terms are given greater emphasis. The *weight* of a term is calculated as

  $$w_t = tf \cdot idf \tag{2.1}$$

  $$idf = \log\left(1 + \frac{N}{n}\right) \tag{2.2}$$

  where w_t is the relevance weight assigned to a document due to query term t, N is the number of documents in the collection, and n is the number of documents containing the term t. Inverse document frequency (*idf*) determines how common a term is in the entire collection. Terms that occur in fewer documents are given higher weighting since they have greater discriminating power [Witten et al., 1999]. Term frequency (*tf*) considers how frequently a term occurs in a particular document and assigns a greater weight to terms occurring more frequently. Finally, length normalisation is used to reduce the adverse impact of longer documents on search results. Longer documents by virtue of their size are more likely to be relevant to many different queries because

they are more likely to contain a variety of terms, otherwise known as the "scope hypothesis". Alternately, they may have the same scope as a shorter document, but simply use more words, known as the "verbosity" hypothesis [Robertson et al., 1993].

- **Cosine Similarity**

  Cosine Similarity operates on a representation of queries and documents as vectors in a multi-dimensional space, where each term is represented in its own dimension. It then measures the angle between the query vector and each document vector in the collection [Luhn, 1957; Salton, 1971; Salton and McGill, 1986; Salton and Buckley, 1988; Witten et al., 1999; Baeza-Yates and Ribeiro-Neto, 1999]. The document vector that has the smallest angle with the query vector has the highest similarity. The cosine measurement is used to normalise the angle to a similarity score between 0 and 1. When the similarity is lowest, the angle is at 90°, and the score is 0. When the similarity is highest, the angle is at 0°, and the score is 1.

$$\text{similarity}(Q, D) = \frac{\sum_{i=1}^{m} w_{qi} \cdot w_{di}}{\sqrt{\sum_{i=1}^{m} (w_{qi})^2 \cdot \sum_{i=1}^{m} (w_{di})^2}} \tag{2.3}$$

  where there is a vector of m terms, w_{qi} is the weight w of term t_i in query Q, and w_{di} is the weight w of term t_i in document D. The weight of a term indicates the degree to which the term discriminates documents from each other.

- **Okapi BM25**

  Okapi Best Match 25 is a probabilistic model that measures the probability that a document is relevant to a given query [Robertson and Jones, 1976; Robertson and Walker, 1999; Jones et al., 2000]. Documents and queries are represented as binary vectors, where a 1 or 0 indicates whether a particular term occurs in the query or document. The metric then determines the probability that a given document is relevant and irrelevant to a query. The function also includes *tf*, *idf*, and length normalisation to improve performance in the case where no relevance information is available.

$$w_t = tf_d \times \frac{\log(\frac{N-n+0.5}{n+0.5})}{k_1 \times ((1-b) + b \times \frac{dl}{avdl}) + tf_d} \tag{2.4}$$

where w_t is the relevance weight assigned to a document due to query term t, tf_d is the terms frequency in the document, N is the total number of documents, n is the number of documents containing the query term, dl is the length of the document, while $avdl$ is the average document length. $k1$ and b are constants. The constant $k1$ affects the strength of the relationship between weight and term frequency, while b alters the impacts of document length.

- **Language Modelling**

  Language Modelling builds a language model for each document in a collection, then for each query, it estimates the probability that the query was constructed from the language model for each document [Ponte and Croft, 1998]. A simple metaphor would be to select ten terms at random (with replacement) from a document and then trying to identify which document they were taken from by analysing the selected terms. Language modelling assumes that users would select query terms that are likely to occur only in relevant documents. The model is slightly more complicated than this explanation since it also has a certain probability of selecting terms from other documents, to consider the distribution of a term across relevant and non-relevant documents. Furthermore smoothing is applied to compensate for data sparseness [Zhai and Lafferty, 2004]. The likelihood of a query given a document is

$$P(Q|D) = \prod_{q \in Q} P(q|D) \tag{2.5}$$

  where Q is the query, D refers to a language model estimated from the corresponding single document. The probability that a query term was generated by the language model of a document is

$$P_\lambda(t|d) = (1-\lambda)\frac{tf_d}{|d|} + \lambda\frac{tf_c}{|c|} \tag{2.6}$$

  where $P_\lambda(t|d)$ is the probability P that query term t was generated by the language model of document d, using the Jelinek-Mercer method of smoothing, λ is a parameter

controlling the probability mass assigned to unseen words, tf_d is the term frequency in document d, $|d|$ is the number of terms in document d, tf_c is the term frequency in the collection c, and $|c|$ is the number of terms in the collection.

Query-Independent Ranking

Query-independent ranking schemes typically rely on link graph information to determine document importance, authority, or relevance [Henzinger, 2001]. They utilise the fact that documents on the Web are linked together by anchors added by document authors and that this gives the Web a structure that can be exploited and from which we can infer relationships [Davison, 2000; Broder et al., 2000a;b]. They are typically combined with document ranking functions to produce an overall rank [Brin and Page, 1998; Sullivan, 2003a]. Furthermore they are used extensively to adjust crawl ordering to improve collection freshness and importance [Cho et al., 1998; Baeza-Yates et al., 2005]

- **Hubs and Authorities (HITS)**

  Hubs and *authorities* is a method for ranking query results in a hyperlinked environment [Kleinberg, 1999]. Authorities are documents that are linked to by many hub documents, and hence are considered important by many different users, a concept very similar to both inlink and PageRank [Page et al., 1998], which we discuss next. Hubs are documents that link to many different authoritative documents, and are hence considered good sources for finding other important documents. There is circularity in the relationship between hubs and authorities, and so an iterative algorithm is required to determine which are which. While the hubs and authorities scheme is typically applied to a sub-graph of documents returned in response to a user query, and can therefore be considered a query-dependent scheme, in our work we implement a scheme that can be considered a simplified hub method that counts the number of inward and outward links of all documents.

- **Inlink**

  The inlink scheme — also known as backlink or in-degree — ranks documents by the number of incoming links [Cho et al., 1998; Upstill et al., 2003a; Fagin et al., 2003a; Castillo, 2004; Baeza-Yates et al., 2005]. Ranking based on inlink considers each URL

link to a document as a vote by the author of the document that the link originates from. Documents with more inlinks are considered more important by web authors, and so, are ranked higher. The inlink scheme has been used to order a crawl in order to retrieve important documents earlier in the crawl. It has also been compared to PageRank with regard to predicting desirable documents and found in one study to be highly similar in performance despite being computationally less expensive [Upstill et al., 2003a]. Another study, by Fagin et al. [2003a], examines query ranking using inlink in an enterprise search environment finding that it performs poorly compared to other ranking schemes.

- **PageRank**

  The PageRank algorithm [Page et al., 1998], shown in Figure 5.1(d), is similar to the inlink scheme, in that it also uses link graph data to determine document importance. However, while inlink uses simple link counts and a link from any two resources is equally weighted, PageRank weights each link based on the importance of the document from which it originates, and the number of outlinks in the origin document. PageRank essentially models the inlink scheme in a recursive fashion across all resources and is used by the Google search engine [Google, 2008] as part of its ranking strategy. The PageRank algorithm can be described as a web surfer that follows links and on occasion jumps to a random page [Page et al., 1998]. It is moderately complex to compute when compared to the inlink scheme, since each document is initially allocated a PageRank value, and then calculations must be performed iteratively until the PageRank values converge with their true values. In recent years several studies have examined ways of improving the efficiency of the PageRank computation [Haveliwala, 1999; Chen et al., 2002; Kamvar et al., 2003a;b; Broder et al., 2004; 2006].

  The PageRank of document A is given as,

$$PR(A) = (1 - \sigma) + \sigma \cdot \left(\sum_{d \in B_A} \frac{PR(d)}{N_d} \right) \tag{2.7}$$

  where σ is a constant dampening value between 0 and 1, $d \in B_A$ is the documents d that link to document B_A, $PR(d)$ is the PageRank of document d, and N_d is the number N of outward link in document d.

- **DynAMoRANK**

  DynAMoRANK is a dynamic application of the Absorbing Model [Amati et al., 2003]. It combines content and link analysis and is produced by a random walk on a modified Markov chain that has been extracted from the Web. Amati et al. [2003] show that it outperforms PageRank on the TREC WT10g and .GOV collections.

- **Static Utility Absorbing Model**

  The Static Utility Absorbing Model (SUAM) also uses the Absorbing Model to gather information from hyperlinks between Web documents [Plachouras et al., 2003]. SUAM determines the utility of documents with regard to how well they allow users to browse their vicinity. That is, it determines how well linked a document is to other documents, thus facilitating user accessibility.

- **TrafficRank and HOTS**

  The TrafficRank and Hyperlink Object Temperature Scale (HOTS) schemes model the Web using an entropy maximisation procedure [Tomlin, 2003]. The TrafficRank scheme ranks nodes by the amount of expected traffic through them, while the HOTS scheme determines the local temperature. Both measures are influenced by the number of inward and outward links.

- **Weighted Link Rank**

  Weighted Link Rank (WLRank) ranks documents by considering various link attributes such as the location of the tag in the HTML structure, the length of the anchor text, the relative position of the link in the document, and the probability of reaching a document while randomly surfing [Baeza-Yates and Davis, 2004]. The results show WLRank can improve precision, with the length of the anchor text appearing to be most effective, while the relative position of a URL on a document is ineffective.

- **Discovery Date**

  The discovery date of a document identifies the time taken to discover a document during a breadth-first crawl beginning at the root document. The sequence of times provides an approximation of the hyperlink graph distance from the root document to

each document [Fagin et al., 2003a]. Documents that have a smaller graph distance to the root are more accessible and hence considered more important. Fagin et al. [2003a] find that discovery date had a significant effect on recall when combined with document ranking functions.

- **HostRank and DirRank**

  HostRank and DirRank are variations of the PageRank algorithm that attempt to reduce the effect of link spamming or farming, a technique where an author creates many pages that all link to a single page, thereby increasing its PageRank score. The HostRank method alters the random surfer model so that instead of periodically jumping to a random page, the surfer jumps to a small set of "trusted pages" [Eiron et al., 2004]. The DirRank method breaks domains by directory structure so that pages that are in the same directory are grouped together when calculating PageRank scores [Eiron et al., 2004].

Query-Dependent Ranking

Query-dependent ranking schemes operate on a subset of a collection produced by a query or accumulated query statistics.

- **WebQuery**

  The WebQuery system examines the link structure of the results returned in response to a query and ranks them by their connectivity [Carrière and Kazman, 1997]. Documents that are more highly linked are given higher ranking. Furthermore, documents that are highly linked by returned results but are not returned by the original query are also ranked highly.

- **Improved Hubs and Authorities**

  Bharat and Henzinger [1998] present methods for improving Kleinberg's hubs and authorities scheme. They observe that the original scheme suffers from three problems:

  1. Mutually reinforcing relationships occur between hosts when one site has a large set of documents pointing to a single document on a second site. This increases

the hub score of the originating site and increases the authoritative score of the second site, giving undue weight to the opinion of a single author.

2. Automatically generated links are given the same weight as links generated with human opinion.
3. Non-relevant nodes found in neighbourhood graphs that are well connected lead to topic drift.

- **ARC**

  The Automatic Resource Compiler (ARC) is a system that compiles authoritative web resources on a given broad topic, similar to those provided by web directories, by adapting Kleinberg's hubs and authorities scheme, and incorporating terms near anchor text [Chakrabarti et al., 1998].

- **Topic-Sensitive PageRank**

  While the original PageRank scheme does not consider whether links are related by topic, topic-sensitive PageRank is an alternative method that computes a set of PageRank vectors that are biased towards different topics [Haveliwala, 2002]. The search results are ranked by a PageRank score that is biased towards the topic of the query terms.

- **Words-in-URL**

  The words-in-URL approach gives a slight preference to documents that contain query terms as a substring in the URL, and is applied once documents are ranked by the likelihood that they are relevant to a query. Fagin et al. [2003a] show that using words in the URL in conjunction with traditional document ranking functions can produce excellent improvements in effectiveness.

- **Hilltop**

  Bharat and Mihaila [2000] describe the hilltop algorithm, which computes a list of the most relevant "expert" pages on a query topic, and determines which pages in a query result set are linked to by this set of expert pages. Pages are then ranked by the number and relevance of expert pages that link to them.

While ranking schemes are primarily used to order search results, they can also by used to order web crawls. Next, we discuss some simple crawl ordering schemes that have been used in the past.

2.2.2 Crawl Ordering Schemes

An important step during the crawling process is managing the order of the documents in the crawl frontier. For crawling to be efficient, we must rank documents in the list such that those that have changed or are popular are more likely to be crawled than those that are static and unpopular. This is analogous to ranking documents in response to a query, but rather than aiming for documents relevant to the query at the front of the list, we want changed, popular documents. While we investigate many crawl ordering schemes later in Chapter 5, here we provide a brief outline of some of the more common crawl ordering schemes that have been used in the literature.

- **Breadth**

  The BREADTH ordering scheme crawls a domain by beginning at the root, retrieving all documents at that link depth level, before any documents at the next link depth level. Breadth-first ordering was used in early crawlers [Pinkerton, 1994], and has been shown to perform reasonably well in retrieving important resources early in a crawl [Cho et al., 1998; Najork and Wiener, 2001], where importance is measured using the PageRank algorithm [Page et al., 1998]. While Najork and Wiener [2001] show that breadth-first ordering retrieves documents with high PageRank early in a crawl, they do not compare it to any other schemes. Other studies have shown that breadth-first ordering performs well during the beginning of a crawl [Castillo, 2004] .

- **Depth**

  The DEPTH ordering scheme crawls a domain by beginning at the root, retrieving all documents accessible via a particular link first, before moving onto the next link in a document. Depth-first ordering has been used for focused crawling [De Bra and Post, 1994], in an attempt to retrieve documents that are most relevant to a given topic first.

- **PageRank**

  The PageRank [Page et al., 1998] ordering scheme, described in Section 2.2.1, has been shown to perform well in retrieving important documents early in a crawl, albeit where the measure of importance is also PageRank [Cho et al., 1998; Castillo, 2004].

The final stage of the querying process is to return an answer set to the user, which we discuss next.

2.2.3 Answer Sets

Once a query has been processed and the documents are ranked by relevance to the user query, an answer set must be created, sorted, and returned to the user, along with a "snippet". While typical sorting algorithms require $N \log N$ comparisons to sort N records, a much simpler approach is to use a max-heap, since only a small subset of documents is returned per page of results, and it is only this subset that needs to be sorted [Witten et al., 1999]. Once the ranked relevant documents are determined and ranked, snippets are created. Snippets contain summaries that are selected from the ranked document, and allow the user to make a more informed judgement of the relevance of the document. These snippets are created from the locally indexed collection and typically, are query-biased, containing passages of text that have a high frequency of the user's query terms [Tombros and Sanderson, 1998]. The URLs and snippets of each document are then presented to the user, typically as ten results per page. Since the snippet is created from the locally cached copy of a document, it is likely to appear to be relevant to the user query. However, once the user clicks on the URL, they are presented with the live Web copy of the document, and this may have changed since the document was last crawled, and therefore, may no longer be relevant.

In this section we have discussed the querying and crawling process, highlighting their interaction, presented various methods for ranking query results, examined various simple crawl ordering strategies and finally, illustrated the creation of answer sets. In the next section we identify methods used by crawlers to update local copies of web collections.

2.3 Collection Update

Whenever data is replicated, issues regarding maintaining consistency and coherency arise. Collections must be kept consistent with the source data, either through a cooperative *push* model, or an independent and autonomous *pull* model.

2.3.1 Push vs. Pull Model

A push model is typically used in cases where strict consistency is required, such as replicated or distributed database systems. In this case, the source notifies replicated collections of changes as they occur. In such cases maintaining consistency and coherence is not difficult.

Alternatively, a pull model is used in decentralised collections, where strong consistency is not possible or costly, such as the Web. In this case, the source must be "polled" periodically, to determine whether changes have occurred in the collection.

Much of the past work on maintaining consistency of replicated data has concentrated on environments where a push model is assumed, particularly database systems [Bernstein et al., 1980; Carvalho and Roucairol, 1982; Krishnakumar and Bernstein, 1991; Pu and Leff, 1991; Golding and Long, 1993; Barbará and García-Molina, 1994; Olston and Widom, 2000; Yu and Vahdat, 2000].

Although there are cases of push-based replication on the Web, such as mirroring, this is typically restricted to systems maintained by a single organisation that has control over both the source and replicated collections [Sivasubramanian et al., 2004]. In the case of replicated web collections that are maintained by crawlers or proxy cache systems, a pull model is applicable. When recrawling web collections using a pull-based model, batch and incremental updates are the two main approaches used.

2.3.2 Batch Update

The batch or periodic update approach does not recrawl any documents more than once during any crawl period. That is, all documents must be recrawled before any can be recrawled again. This approach is used by Google when conducting a deep crawl on a monthly basis [SEO Today, 2002; Googlerank.com, 2005]. The batch update method ensures that all documents are retrieved within a particular time period, however, it does not effectively

maintain the freshness of documents that change more frequently than the update period [Edwards et al., 2001].

Consider the following simple example: if a search engine must maintain a repository of documents and it takes less than one week to crawl all the documents, it could conduct a batch update on a weekly basis. While this would guarantee that no document was more than one week old, any documents that change more frequently than once a week would not be up-to-date.

2.3.3 Incremental Update

The incremental update approach does not require that all documents are retrieved before revisiting documents that may have changed. This approach is used by Google to maintain more frequently changing documents [SEO Today, 2002; Googlerank.com, 2005]. While this method does effectively improve the freshness of changing documents, it does not guarantee an upper bound on freshness like the batch update approach.

Typically, large-scale crawlers adopt a combination of both batch and incremental crawling to improve freshness. Earlier approaches to improving freshness have focused on timestamps.

2.3.4 Early Consistency Schemes

Much of the early work on maintaining consistency has focused on the maintenance of files with the File Transfer Protocol (FTP) through the use of timestamps similar to the `Last-Modified` date.

Work by Cate [1992] describes an early system for maintain consistency of local copies of files downloaded from an FTP site. Users specify an *update threshold* which indicates the frequency at which files should be updated, and is expressed as a percentage of the files reported age. The default operation of the system guarantees that files are up to date as of 5% of the reported age. For example, a file with a reported age of twenty days is updated once a day. This approach assumes that older files are less likely to be updated than newer files.

In related work, Gwertzman and Seltzer [1996] compare various early caching consistency algorithms using trace-driven simulations. They show that the weak cache consistency scheme

used by the Alex FTP cache systems [Cate, 1992] reduces network and server load more effectively than a TTL or invalidation protocol.

While these early schemes were concerned with FTP sites, later work concentrated on change of web documents, particularly HTML files. In the next section we discuss methods for evaluating the performance of search engines and crawlers.

2.4 IR Evaluation Metrics

In this section we provide a brief discussion of the various information retrieval (IR) metrics used to evaluate the performance of search engines.

2.4.1 Precision and Recall

In IR the performance of a search engine is evaluated through the use of the measures *precision* and *recall* [Grossman and Frieder, 1998; Witten et al., 1999]. Suppose there is a set of documents D in a collection and a set of queries Q. Suppose also that a user has evaluated the relevance of each document to each query, and so we have a subset of documents $d_i \subset D$ that are evaluated as relevant to query $q_i \in Q$. Finally, suppose that we have a returned set of documents d_r at cutoff r. We can then evaluate the performance of a search engine by determining the percentage of relevant documents it returns in response to query q_i after examining the first r returned documents.

$$P_{ir} = \frac{|d_r \bigcap d_i|}{|d_r|} \tag{2.8}$$

$$R_{ir} = \frac{|d_r \bigcap d_i|}{|d_i|} \tag{2.9}$$

The precision P_{ir} of results at cutoff r for query i is the fraction of returned documents that are relevant in the top r returned documents, while the recall R_{ir} of the results at cutoff r for query i is the fraction of the total relevant documents in the collection that were returned in the top r returned documents. For example, if 100 documents are retrieved and 40 of them are relevant, the precision at 100 (P_{100}) is $40/100 = 40\%$. If the collection contains a total of 80 relevant documents, the recall at 100 (R_{100}) is $40/80 = 50\%$.

The recall and precision scores can be summarised using many different measures.

- *Precision-at-10* (P_{10}), is the precision after ten answers have been inspected, that is P_{10}. This measure is particularly suited to quantifying search engine effectiveness since most search engines return only ten results per page of results by default [Google, 2008; Yahoo!, 2008; Ask.com, 2008; Altavista, 2008; Lycos, 2008] and many users do no look past the first page of results [Jansen et al., 1998; Joachims, 2002; Lempel and Moran, 2003; iProspect, 2006]. Having an arbitrary cutoff of ten does not average well across different queries particularly when the number of relevant documents in a collection varies. A query with only three relevant documents can only achieve a maximum P_{10} of 0.3, while a document that has several thousand relevant results is likely to have a very high P_{10} score regardless of which scheme is used.

- *Average Precision* (AVP) averages the precision values after each relevant document is retrieved. This is typically averaged over several queries to produce a *Mean Average Precision* (MAP) score. Unlike many other measures, average precision requires no thresholds or interpolation, and the contribution of ranked relevant documents is intuitive, that is, a highly ranked relevant contributes more than a lowly ranked one. Furthermore, average precision is extremely sensitive to changes in the rank of relevant documents, the contribution of each relevant document is known, and finally it is robust in practice, with comparable results for different collections. However, average precision also has weaknesses. First, there is no user application that directly motivates MAP, and hence, it is not an application measure, instead it is more an overall system evaluation. Second, it is not favoured by statisticians, since relevant documents affect the score of all relevant documents ranked below it [Voorhees and Harman, 2005].

- *Eleven-Point / Three-Point Average Precision* averages the precision values at either each of the eleven or three recall intervals from 0% to 100%, that is 0, 10, 20 ... 80, 90, 100 in the case of eleven-point precision, and 20, 50, and 80 with three-point precision. In effect, three-point precision determines the precision score after 20%, 50%, and 80% of all relevant documents are retrieved and then averages these scores.

As defined earlier, $d_i \subset D$ is the subset of documents that are judged as relevant to query i, $P_{(r)}$ is the precision at cutoff r. Let k_j reference the j-th element in the set $k = \{0.0, 0.1, 0.2 \ \ldots \ 0.8, 0.9, 1.0\}$. Then eleven-point average precision is calculated as,

$$11PP = \frac{1}{11} \sum_{j=1}^{11} P_{(k_j \times \ |d_i \subset D|)} vspace5pt \tag{2.10}$$

Let k_j reference the j-th element in the set $k = \{0.2, 0.5, 0.8\}$. Then three-point average precision is calculated as,

$$3PP = \frac{1}{3} \sum_{j=1}^{3} P_{(k_j \times \ |d_i \subset D|)} \tag{2.11}$$

Both eleven-point and three-point average precision suffer from a problem where a small difference in retrieval could lead to a large difference in the score. For instance, when measuring three-point average, if the 0.2 recall level is not reached, the score will be 0. Consider the example where there are 300 relevant documents in a collection and 59 of the 100 documents returned in response to a query are relevant. The 3 pt precision will be 0.0, whereas if 60 relevant documents were returned, the 3 pt precision would be 0.2 (0.6/3).

- *R-precision* selects a cutoff that is equal to the number of relevant documents in the collection, instead of an arbitrary cutoff. This makes the results from various queries comparable, and hence their average is meaningful, unlike precision-at-10. Furthermore, *R-precision* is more of a measure of overall system performance, compared to precision-at-10, which is more applicable to a particular application, such as the number of relevant results on the first page of results. R-precision is calculated as the precision after r documents are retrieved, where r is the number of relevant documents in the collection,

$$\text{R-Precision} = P_{(|d_i \subset D|)} \tag{2.12}$$

- *Mean Reciprocal Rank* (MRR) determines the inverse of the rank of the first relevant answer, averaged over several queries. It is typically used when evaluating the performance of question answering systems, page finding, and one class classification problem, where typically there is only one correct answer and the aim is to return this as the highest ranked result.

2.4.2 Rank Correlation Metrics

Rank correlation metrics can be used to evaluate changes in the rank of search engine results. Typically, the ranked results produced by a particular scheme are compared to that of another scheme to determine the amount of disorder in the ranks [Fagin et al., 2003b]. While there are many different correlation schemes, not all are suitable for ranked lists. An important consideration when evaluating ranked lists is that changes in the order of highly ranked items are more important than changes to lower ranked items. While both the *Spearman rank correlation coefficient* and em Kendall's Tau can measure changed in the order of a list, they do not consider the rank of items.

- *Spearman Rank correlation coefficient* determines differences in the order of two lists and produces a score in the range -1 to 1. The magnitude of the score indicates the strength of the correlation, that is, a score that is closer to either 1 or -1 indicates that there is a stronger correlation, while a score closer to 0 indicates that there is a weaker correlation. The sign indicates the direction of the correlation. A positive score indicates a positive relationship, that is, as ranks in one set increase, so do the ranks in the second set. A negative score indicates that as ranks in one set increase, the ranks in the second set decrease [Spearman, 1904].

$$\text{Spearman Rank} = 1 - \frac{6 \sum \phi_i^2}{\varphi(\varphi^2 - 1)} \tag{2.13}$$

  where ϕ_i is the difference in rank for each corresponding item in two lists, and φ is the number of item pairs.

- *Kendall's Tau* indicates the degree of similarity between two lists by determining the number of inversions of pairs necessary to transform one ranked list into another [Kendall and Gibbons, 1990].

$$\tau = \frac{4F}{z(z-1)} - 1 \tag{2.14}$$

  where z is the number of items, and F is the sum of items ranked after the given item by both rankings.

- *Dissim* determines the degree to which two lists differ by allocating each item in a list a score calculated by the difference in its rank in the second list. Items that have the same rank in both lists contribute a score of zero, while items that have changed contribute an amount determined by the inverse of their rank [Moffat et al., 2006]. These scores are summed and then normalised by the maximum score, achieved when all items do not appear in the second ranked list.

$$\text{Dissim Contribution}(y) = \left| \frac{1}{T + x_y} - \frac{1}{T + \ell(x_y)} \right| \tag{2.15}$$

  where T is a damping factor, x_y is the position of item y in one ranked list, and $\ell(x_y)$ is the position of the same item in the second list.

In this section we have introduced various IR evaluation metrics, which we will adapt into crawler evaluation. In the next section we identify the various crawler evaluation metrics.

2.5 Crawler Evaluation Metrics

In the previous sections we defined some of the standard IR evaluation metrics. Crawler evaluation has had a variety methods defined. To compare crawl ordering schemes it is necessary to define crawler performance metrics. The sole aim of a crawler is to find new and changed documents. Subsequently, crawler performance is measured with a freshness metric that determines whether a document has been recrawled, and whether or not the document has changed or new [Cho and García-Molina, 2000a]. We discuss document change in Section 2.5.1. One method for measuring crawler performance that has been used before is the "ChangeRatio" metric [Cho and Ntoulas, 2002], which is similar to the metric used by Edwards et al. [2001] to evaluate incremental crawling techniques.

- **ChangeRatio**

  The *ChangeRatio* metric detects the percentage of downloaded items that have changed. For instance if 1000 documents are downloaded and 600 of those documents have changed, the *ChangeRatio* is 0.6 (60%). When more downloaded items have changed, the scheme is considered more successful and the *ChangeRatio* score increases. To avoid variations in results over different cycles the *ChangeRatio* is averaged over several crawl

cycles. Cho and Ntoulas also note that there may be cases where items vary in "importance", which they account for by assigning weight w_i to each item o_i. Furthermore they define $1(o_i)$ as an indicator with a value of 1 when item o_i has changed and 0 when it has not. While they define *ChangeRatio* as shown in Equation 2.16, in practice they use the measure shown in Equation 2.17.

$$ChangeRatio = \sum_{i \in R} w_i \cdot 1(o_i) \tag{2.16}$$

$$ChangeRatio = \frac{1}{|R|} \sum_{i \in R} w_i \cdot 1(o_i) \tag{2.17}$$

- **Freshness and Age**

  Cho and García-Molina [2000a] define the freshness and age metrics as

$$F(o_i; t) = \begin{cases} 1 & \text{if } o_i \text{ is up-to-date at time } t \\ 0 & \text{otherwise} \end{cases} \tag{2.18}$$

  where up-to-date means that the local copy of a downloaded item matches the source item. They define freshness of the entire collection as

$$F(U; t) = \frac{1}{|U|} \sum_{o_i \in U} F(o_i; t) \tag{2.19}$$

  where U is the set of all locally stored items. Freshness measures the fraction of locally stored items that are up-to-date.

  They define *age* of item 0_i at time t as

$$A(o_i; t) = \begin{cases} 0 & \text{if } o_i \text{ is up-to-date at time } t \\ \text{t} - & \text{modification time of } o_i \text{ otherwise} \end{cases} \tag{2.20}$$

  and the age of the collection as

$$A(U; t) = \frac{1}{|U|} \sum_{o_i \in U} A(o_i; t) \tag{2.21}$$

  The *age* indicates how "old" the local copy of an item is. Under ideal conditions, a collection will have a high normalised *freshness* value of 1 and a low normalised *age* value of 0.

	Document changed or new	Document not changed
Crawler Revisits	SUCCESS	FAILURE
Crawler Does Not Revisit	FAILURE	SUCCESS

Table 2.1: Matrix for determining the success or failure of a crawl in acting on a single document. In this case retrieving new documents is considered a success.

- **Divergence**

  Divergence is a staleness metric introduced by Olston and Widom [2002] that represents the amount of difference between a local copy of an item and the source item. When there is a large divergence, there are more differences between the local copy and the source item, and so, the scheme is less effective in maintaining collection freshness.

Our measure, "hit rate", while similar to ChangeRatio, contains several differences. First, while Cho and Ntoulas only measure the ChangeRatio at the end of a crawl, we measure "hit rate" after each document is retrieved, hence providing a clearer indication of a scheme's performance. Second, we consider new documents as part of our evaluation.

When the crawler retrieves a document that has changed or is new, this is considered a success; conversely, retrieving a document that is the same as the copy already indexed is considered a failure. This can be broken down into the confusion matrix shown in Table 2.1.

Similar to the definition of *precision* and *recall*, we define "*hit rate*" and "*crawl rate*" at cutoff r.

$$h(r) = \frac{\text{Number changed or new documents found at cutoff r}}{\text{r}} \tag{2.22}$$

$$c(r) = \frac{\text{Number changed or new documents found at cutoff r}}{\text{Total changed documents in collection}} \tag{2.23}$$

The *hit rate* h(r) metric measures the percentage of changed or new documents in the crawled collection after r documents have been retrieved, hence determining how accurately the crawl scheme locates changed and new documents. The *crawl rate* c(r) metric measures the percentage of changed or new documents in the collection that have been retrieved after r

documents have been retrieved, hence determining how thoroughly the crawl scheme locates change and new documents.

We then plot the performance of a scheme as the *hit rate* against the *crawl rate*, the *hit rate* against the percentage of the collection crawled and the *crawl rate* against the percentage of the collection crawled.

As stated earlier, crawler evaluation determines whether changed documents are re-crawled. This therefore relies on a method of detecting change.

2.5.1 Measuring Document Change

The metrics in the previous section that evaluate crawler effectiveness require a definition of change. In this section we present various methods for detecting and measuring change and briefly compare them.

Simple Change Detection

A simple method of detecting byte-wise change between two documents is to compare a hash value, such as that computed by an MD5 hash [Rivest, 1992]. If two documents have the same hash value it is extremely likely that they are identical. Hashing has been used in many web change, collection freshness, and mirroring studies [Wills and Mikhailov, 1999a; Bharat and Broder, 1999; Mogul, 1999; Wills and Mikhailov, 1999b; Cho and García-Molina, 2000c; Brewington and Cybenko, 2000b;a; Cho and García-Molina, 2003a]. While hashing is fast and simple to compute, it is not a useful definition of change from a crawling perspective since it does not quantify the differences between two versions of a document, instead producing a binary indication of change regardless of the size or type of that change.

Detecting Meaningful Change

The size and type of change in a document may be important when assessing crawler effectiveness. Some changes are unlikely to alter the index term entries for that document, and hence not affect user query results. For example, changes in formatting such as the addition of a "`<hr>`" HTML tag to a document will not affect results to queries for most search mechanisms. Furthermore, the alteration of an advertisement within a document is also unlikely to affect search results. Many studies of change on the Web have shown that changes are

(A) the quick brown fox jumped over the lazy dog

(B) the lazy dog jumped over the quick brown fox.

the quick brown	**quick brown fox**	brown fox jumped	fox jumped over
	jumped over the	over the lazy	
the lazy dog	lazy dog jumped	dog jumped over	over the quick

Figure 2.2: Measuring the difference between sentence (A) and (B) using a three word shingling metric. The shingles in **bold** *are common to both sentence (A) and (B).*

localised, due to rotated banner advertisement, changing date fields, and so on [Wills and Mikhailov, 1999a; Ntoulas et al., 2004a]. It would be a waste of crawler resources to recrawl these documents because doing so is unlikely to result in important changes to the index.

A first criterion for a meaningful measure of change, therefore, is that it should only take into account HTML content that is indexed by the search engine.

Shingling

The shingling scheme [Broder et al., 1997] measures the *resemblance* of two documents by removing all HTML from the documents, converting their text to lowercase and then measuring the number of unique sequences of n words, known as n-grams or n-shingles, that the two documents have in common, as shown in Equation 2.24. The scheme can also be used to measure the *containment* of one document within another. For example, Equation 2.25 computes the percentage of document (A) that originates from document (B).

$$\text{Resemblance} = \frac{\text{Number of } n\text{-shingles (A) and (B) have in common}}{\text{Total number of unique } n\text{-shingles in (A) or (B)}} \tag{2.24}$$

$$\text{Containment} = \frac{\text{Number of } n\text{-shingles (A) and (B) have in common}}{\text{Number of unique } n\text{-shingles in (A)}} \tag{2.25}$$

The example in Figure 2.2 uses three word shingles and produces ten unique sequences, of which four are common to both sentences, producing a similarity of $4/10 = 40\%$.

Since shingling requires a large amount of processing and storage, sketches are used, which store a specific number of shingles selected from the document. The selection criterion

may be positional, frequency-based, or structural and has been surveyed by Hoad and Zobel [2003]. These are converted into distinct numbers that are then sorted. "Supershingles" can be produced by creating shingles of the sorted sketches.

Other Measures of Change

Various measures of change have been used as part of large scale web document monitoring and reporting tools. These tools allow users to specify web documents that they want monitored, and the type of changes that they want reported. The WebCQ system [Liu et al., 2000], in particular, allows the following types of change to be detected:

- Content update: Any update on a document
- Content insertion: Increase in size (above a threshold)
- Content deletion: Reduction in size (above a threshold)
- Link change: New or removed HTML links
- Image change: New or removed images
- Word change: New words added or existing words removed
- Phrase update: Detect changes to specific phrases
- Phrase deletion: Detect removal of specific phrases
- Table change: Detect changes to specific tables
- List change: Detect changes to specific lists
- Arbitrary text change: Identify any changes in specified fragments
- Keyword: Detect addition or removal of selected keywords

We have evaluated several of these schemes in our work on change metrics in Section 4.3, and investigated these in a web crawling and search context.

ChangeDetector, another tool for monitoring web sites for change, uses document classification to filter changes by topic and uses entity-based change detection to filter changes by semantic concepts, such as names and dates [Boyapati et al., 2002]. One such example is the presence of a *date* near a *location*, which is indicative of a press release.

When detecting change, the ChangeDetector scheme first looks for byte-wise differences. If a change is found, the scheme then processes the document with an XML-aware difference algorithm followed by entity extraction. The difference algorithm breaks the document into nodes based on the W3C Document Object Model [Hégaret et al., 2005]. The text that falls below a node is normalised by ignoring differences in whitespacing and structure. If a hash of the text detects differences, the algorithm attempts to align the text with *insertions* and *deletions*. Any "uninteresting" changes, such as spelling corrections, are ignored. The remaining changes are filtered through entity extractors to detect different types of semantic content. The distance between entities is recorded into a database allowing users to submit queries.

Flesca and Masciari [2003] present a change metric that has been implemented in CMW, a change detection system that can create web update triggers that allow users to identify changes they want monitored and the actions to perform when they occur. Their approach does not examine the exact sequence of changes that are required to produce the modified document, but instead the amount of change that has been made. They represent documents as trees and compare document subtrees to determine the amount of difference between them.

Work by Chawathe and García-Molina [1997] examined change detection in structured text. Their scheme MH-DIFF, detects change in the hierarchy of a tree using the insert, delete, update, copy, move, and glue operations to classify changes.

- Insert adds a new node to the hierarchy.
- Delete removes a node from the hierarchy.
- Update alters the label of a node.
- Copy creates a copy of a subtree in another position in the hierarchy.
- Move removes a section of a subtree and places it into another position in the hierarchy.
- Glue replaces a section of the subtree with a copy of another subtree in the hierarchy.

While there has been a significant amount of research dealing with measuring change, particularly in web documents, much work dealing with modelling the Web has dealt with simple change metrics.

2.6 Poisson and Simple Change Measures

There has been a variety of work that has examined ways of improving crawler efficiency and collection freshness on the presumption that the Web changes according to a Poisson process. Much of this work has used simple measures of change based on hashing. A Poisson process is used to model a set of events that occur *randomly* and *independently* at a *fixed rate* over time [Taylor and Karlin, 1998]. Poisson processes have been used to model events over time, such as the number of times a web server is accessed per minute, and the number of accidents at an intersection each month.

We first examine the various studies by Cho et al. using a Poisson process to model web change.

2.6.1 Work by Cho et al.

In work examining various synchronisation policies and their effectiveness in keeping a collection up-to-date Cho and García-Molina [2000a] assume that web change follows a Poisson process. They examine various ordering policies (fixed, random, purely random), uniform allocation versus non-uniform allocation policies, and synchronisation frequencies. They examine the following synchronisation ordering methods:

- Fixed order: Visits URLs in a list repeatedly in the same order.
- Random: Maintains only the URL of the root document of a site with recrawls beginning from the root document. The link structure may change between crawls.
- Purely random: Synchronises documents on demand, as they are requested by a user.

The also test the following resource allocation methods:

- Uniform allocation policy: Synchronises all elements uniformly at the same rate.
- Non-uniform allocation policy: Synchronises elements proportionally. Elements that change more often are revisited more often.

Furthermore, they define freshness as the number of elements (documents) that are up-to-date (where the local document is the same as the real world document) and age as the sum of the difference between the current time and the modification time of each

element in the database. Their results show that the fixed-order policy performs better than both the random and purely random policies for maximising collection freshness and reducing collection age. They also find that the uniform allocation policy is always more effective than the proportional policy when the goal is to maximise the freshness of the database (averaged over time) and the amount of change is not considered.

They also argue that documents that change very frequently or infrequently should not be revisited often. Documents that change infrequently do not need to be revisited often by definition, and they argue that documents that change very frequently will not remain up-to-date for long once they are refreshed.

In contrast to our work, they do not consider the significance of changes or the popularity of the document to which changes are made. The home-pages of many sites are likely to change very frequently, while also being of great importance to users. For example, over a four year period the RMIT University home-page has changed on average over 12 times per month [Internet Archive, 2008], however this page may not necessarily be updated by their schemes. Furthermore, their work requires a long history of change to accurately predict change frequency. Cho and García-Molina [2003a] have also shown that accurately estimating frequency of change is difficult unless there is a long change history, or the Last-Modified date is available.

In other work, Cho and García-Molina [2003b] examine the effectiveness of different web document refresh schemes, again assuming that changes follow a Poisson process. They introduce age and freshness metrics [Cho and Ntoulas, 2002], which we discuss further in Section 2.7.

Cho and García-Molina examine proportional and uniform synchronisation policies. The proportional policy refreshes resources at a frequency that is proportional to the frequency with which they change, while the uniform policy synchronises all resources at the same rate.

Cho and García-Molina also examine synchronisation order, comparing fixed order, random order, and a purely random ordering.

Mathematically they show that fixed order maintains the highest freshness and lowest age, when assuming that all items change at the same rate. They also show that the uniform policy is superior to the proportional policy, since the resources remain up-to-date for longer when updating items that change less frequently. They state that this is not due to their

discrete freshness metric but have no formal proof. They show that, to improve freshness, items that change too frequently should be penalised. However, to improve age, they show that items that change more frequently should be revisited more frequently and produce an optimal solution for this.

Cho and García-Molina also study weighted freshness to try and incorporate "importance", showing that items that have a higher weight should be revisited more frequently, however the frequency is not proportional to the weight.

Cho and García-Molina verify experimentally that popular documents change according to a Poisson process, by examining the change frequency of 720,000 documents from 270 popular sites crawled on a daily basis for four months. They show that the optimal policy is significantly better than the uniform and proportional policies (500% freshness improvement and 930% age improvement over proportional). Finally they show that the optimal scheme is more useful when documents change more frequently. While these results are substantial, the use of a simple discrete freshness metric in conjunction with simple change metrics would not consider the rate at which significant changes are made to documents. Furthermore, the fact that only popular sites are used in the experiments would tend to skew the results. Their results do not consider the impact that crawling would have on document ranking. Furthermore, they do not consider the addition of new documents.

In the next section we highlight the work of others in the area of crawler efficiency using a Poisson model of change.

2.6.2 Work by others

In other work that assumes that the Web changes according to a Poisson process, Coffman et al. [1998] examine the problem of search engine coherency from a queuing (theory) background. They relate the problem of revisiting web documents to the multiple-queue single server (polling) system, where they consider the crawler to be the server and the documents to be the stations in the polling system. They formulate a model and determine that the optimal result is achieved when documents are crawled uniformly. However, this work is purely theoretical and does not consider new documents, and so it is unclear how the results would compare with live web data.

While all the work described in this section uses collections assuming a Poisson process for change modelling, there have been studies that have shown that the Web does not change entirely at random, with for example changes occurring more frequently during US working hours [Brewington and Cybenko, 2000a]. Padmanabhan and Qiu [2000] suggest that the update of a news service is not modelled according to a Poisson process.

In the next section we discuss work that has used schemes that try to adapt to change frequencies as they are detected.

2.7 Adaptive Crawl Ordering Schemes

Previously, in Section 2.2.2, we presented some static crawl ordering schemes. In this section we present an overview of work on methods of detecting change and adapting collection updates accordingly. We present work that examines change detection from a crawling perspective, studies that incorporate user feedback as part of crawl ordering, and finally work that has examined adaptive schemes in a data warehousing environment.

2.7.1 Adaptive Crawling

Adaptive crawling schemes attempt to detect changes as part of the crawling scheme and alter the crawl based on this information. One study into adaptive crawling describes the design and implementation of an incremental crawler for the WebFountain project [Edwards et al., 2001].

The model, based on linear programming, considers crawl strategies for improving document coherence and maintaining collection freshness. Unlike much other work — but similar to our work described in Section 4.3 — their design does not make assumptions about the statistical nature of changes made to web documents. Instead it adapts to actual change rates detected as part of the crawling process.

The design implements 256 different change frequency buckets for partitioning documents by change frequency, grouping documents with similar change frequencies together. Documents that change very rapidly are handled separately, on the assumption that these documents are typically media sites.

They find that different objectives produce different optimal solutions, but find a solution that considers several criteria to produce a good overall result.

- When the objective is to reduce the number of obsolete documents that are present in the final time period, the optimum solution is to crawl documents only during the final period, ignoring them during other time periods.
- With an objective to reduce the number of obsolete documents in each time period, the optimum solution is to crawl fast-changing documents in many time periods, while ignoring the 40% of documents with the lowest change rates.
- If the objective is to minimise the number of obsolete documents in the final time period, but to still try to consider the number of obsolete documents in each time period, the optimum solution is to crawl all documents just once during a crawl cycle but spread them across the entire cycle.

In this work, the model considers a document obsolete if it is no longer a match of the version on the Web, as determined by *shingling* [Broder et al., 1997], described in Section 2.5.1. While their results are promising, they are simulated, and they note that it is not possible to run the crawler model for a longer period and obtain a useful mathematical solution. Therefore, their scheme must be periodically reset and start again from the beginning, something they argue would be necessarily anyway in practice in order to update parameters and re-optimise the crawler.

In another major study, Cho and Ntoulas [2002] examine how to effectively use sampling to detect document change. They measure the collection freshness while comparing the following three document crawling methods.

- **Round Robin**

  Round robin downloads a different subset of the collection during each crawl cycle, guaranteeing that all documents are retrieved once each. This method was used by the early version of the Google crawler [Brin and Page, 1998] and the Mercator crawler [Heydon and Najork, 1999].

- **Change Frequency Based**

  The change frequency method uses past change history to determine how frequently documents are changing, and hence how frequently to revisit them. This method is investigated further in other work [Coffman et al., 1998; Cho and García-Molina, 2000a].

- **Sampling-Based**

  The sampling-based method retrieves a small sample from each site to determine what percentage has changed, then allocates the remaining resources accordingly.

In the study they conduct crawls over a collection in cycles, downloading a limited number of documents per cycle. Furthermore, they define several evaluation metrics, though they use only the *ChangeRatio* measure, which we discussed in Section 2.5.

Cho and Ntoulas also define two different sampling policies, *proportional* and *greedy*. Once the initial sample is made, the *proportional* scheme allocates the remaining resources in proportion to the number of changed resources at each site. In contrast, the *greedy* scheme allocates all the remaining resources to the site that had the most changed resources, then if any resources are left they are allocated to the next most dynamic site and so on. While the *greedy* policy is expected to produce a better *ChangeRatio* than the *proportional* scheme when the estimation is correct, it is also expected to have a larger variation in performance, producing a worse result when the estimation is incorrect.

Cho and Ntoulas also examine the optimal sample size for producing the highest *ChangeRatio*. They also present an adaptive sampling process that detects the *ChangeRatio* of each site during the sampling process and examines the confidence intervals. Once the confidence interval is large enough to select a site for crawling, the scheme switches to the *greedy* policy. They note that both the *greedy* and adaptive schemes may never download some documents, and they examine this further in their experimental results.

Cho and Ntoulas examine "subset" sampling when there is only a small volume of resources available. In this case, sites are grouped into subsets and then, in each download cycle, only documents from one subset are sampled and downloaded. Different subsets are visited in a round robin fashion over multiple download cycles. They discuss subset size but do not provide an optimal subset size.

Cho and Ntoulas test these schemes against a web collection of 353,000 documents from 252 web sites, retrieved on a monthly basis for six months. They repeated the data for experiments requiring a longer change history. They also test some of the schemes against synthetic data following a normal change distribution.

Their results show that:

- The *greedy* and *adaptive* schemes perform extremely well, with an average *ChangeRatio* around 75%, compared to around 40% for the round robin, proportional, and frequency-based approaches.
- When the volume of resources allocated to sampling is high, the performance of all schemes is the same since all documents are downloaded. When sampling resources are low, the performance of the *greedy* policy degrades.
- Subset sampling improves performance when there are limited resources.
- With longer change history data the frequency policy gradually improves in performance, requiring about 100 download cycles to match the performance of the *greedy* policy.

To evaluate the *greedy* policies ability to revisit all documents, they introduce a measure of "fairness", which they define as:

$$\text{fairness} = \frac{\#\text{ of changed and visited documents up to the } i\text{th cycle}}{\text{total }\#\text{ of changed documents up to the } i\text{th cycle}} \tag{2.26}$$

Importantly, this measure does not consider multiple missed changes as a negative, as long as a changed document has been visited at least once. After five download cycles, the *greedy* policy visits about 80% of the changed resources at least once. Cho and Ntoulas examine the number of times changed resources are revisited in each change group, and show that the number of visits is proportional to the number of changes. When compared with synthetic data there is a close performance between the schemes with marginal differences, though with the *greedy* and adaptive scheme still perform best.

A different study into the *sampling method* [Cho and Ntoulas, 2002], by Ghodsi et al. [2005], introduces a hybrid method for updating collections. Their work combines the *change frequency* method [Cho, 2001] with their own version of the *sampling method*. They use the sampling method for the initial crawls until a large history is available, then switching to the change frequency method. They also modify the sampling method and instead have an iterative sampling method that downloads a sample size during iterations.

Initially the sample size is equal for each site, however this adapts during iterations, allocating more resources to sites that contain a larger number of changed documents.

Their data set consists of 100,000 documents (1000 web documents from 100 web sites) crawled once a fortnight for eight weeks, a total of four times. They then repeat the four cycles 125 times for a total of 500 cycles, measuring the efficiency of the different schemes against the collection. They measure efficiency as the number of retrieved documents that changed against the total number of retrieved documents.

Their results show that the standard adaptive sampling had an efficiency of 75%, compared to their improved sampling scheme at 81% efficiency, and their hybrid scheme, which had an efficiency of 87%.

2.7.2 User Feedback

Several studies have also considered user feedback as part of an adaptive crawling algorithm.

Bullot et al. [2003], for example, investigate user feedback for crawling. Their work compares various crawl ordering schemes so that popular documents are kept up-to-date. Their approach maintains a queue, with URLs at the front of the queue visited next. The score of any URL that is visited is reset to 0, new documents are inserted into the queue with a high score, and older URLs are given higher scores.

They monitor statistics regarding *category*, *frequency of click*, *position of a URL*, and *next penalisation* to determine which URL to visit next.

- **Category**

  The category of each clicked URL is noted and other URLs belonging to the same category can also be visited. In this way URLs that are not clicked, but on the same topic, also have their score affected.

- **Frequency of Click**

  Frequency of click keeps track of the frequency at which URLs are visited by users. URLs that are visited more frequently by users are rated higher than those visited less frequently.

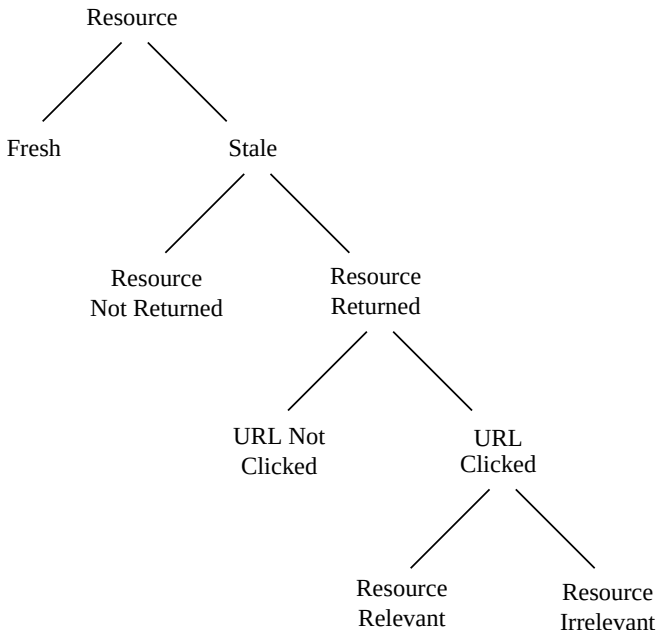


Figure 2.3: The embarrassment metric decision tree.

- **Position of URL**

  Position of URL indicates the location of a URL on the results pages. Since URLs on the results page are sorted by importance, URLs at the top of the first page are more important than any other location on any other pages.

- **Next Penalisation**

  Next penalisation reduces the score of all URLs on pages where users have clicked the next button, since users have most probably found the results either irrelevant or repeated. If a user clicks on some URL before clicking next, URLs on the page are penalised less. If users click on page five instead of next, all URLs on pages one through four are penalised.

Bullot et al. [2003] also discuss the effect of updating score information at different intervals. They propose updating the score information whenever a URL is clicked (real time update), a periodic update at a set interval (scheduled time update) and a hybrid update model (scheduled limited update) that updates whenever a limited number of URL clicks are made. The work is only a proposal and not investigated either experimentally or theoreti-

cally. Furthermore, while they do propose schemes that alter the crawl order with regard to various user feedback, they nonetheless assume that changes follow a Poisson process.

In another study examining user feedback, Schaale et al. [2003] introduce a new method that analyses user queries to determine which documents should be crawled. Their scheme ranks domains based on their relevance to user queries, then combines this with the existing ordering method to produce a new crawl order. The work is untested.

Wolf et al. [2002] introduce the concept of search engine "embarrassment", which measures the likelihood that a user clicks on a search result only to find that it is irrelevant to their query, identifying how embarrassed the search engine would be by the failure of a user's search result. As highlighted in the decision tree shown in Figure 2.3, the embarrassment metric breaks the evaluation into steps. It determines whether a resource is fresh or stale, and in the latter case whether the stale resource is returned in response to a user query. If returned, the scheme determines whether the resource is clicked by the user. Finally it determines whether the clicked stale document is relevant to the query.

They model the probability of a user clicking on a result, based on its rank and page position. They also consider combinations of web update models. Their results show that their scheme for determining the optimal number of crawls for each document outperforms a proportional and uniform scheme. However, their work does not consider cases where the crawler fails to retrieve highly relevant, new documents during the crawl.

Finally, in an empirical study, Pandey and Olston [2005] examine how to schedule the recrawling of documents to improve user experience. They formulate a user-centric search repository quality metric that measures the impact of crawling strategies on users. We investigated a similar concept [Ali and Williams, 2003], which we describe in Chapter 4. They compare three schemes that analyse past change to predict future change and measure their resource usage:

- **Staleness-Based Refreshing**

  The staleness-based approach attempts to reduce the number of stale documents in the search repository [Cho and García-Molina, 2000a]. They use this in conjunction with shingling [Broder et al., 1997] and the transportation algorithm [Wolf et al., 2002] for scheduling.

- **Embarrassment-Based Refreshing**

  The embarrassment-based approach minimises the level of "embarrassment" caused to the search engine [Wolf et al., 2002]. Embarrassment increases when a user clicks on a search results and finds that it is irrelevant. Again, this is used in conjunction with shingling [Broder et al., 1997]. The results are simulated using a Poisson update distribution. Click frequencies are simulated using a Zipf-like function. In the original work by Wolf et al., the likelihood of relevance is simulated by selecting a uniform random number between 0 and 1. Instead, Pandey and Olston assume that a document that undergoes change becomes irrelevant to an average query if the fraction of shingles that change exceeds a given threshold.

- **User-Centric Refreshing**

  They determine user impact by ranking with *tf.idf* and inlink count.

To determine user interest, Pandey and Olston use the AltaVista query set. They allow each scheme to crawl all documents on the initial crawl, then on subsequent crawls only a certain number of documents were allowed to be refreshed. They compared the volume of resources required by each scheme to achieve the same level of collection quality. Interestingly, their results show that a user-centric approach requires substantially fewer resources to achieve the same level of repository quality. This is particularly due to their use of query relevance, which avoids crawling documents that change in "uninteresting" ways.

It is important to note that in their work, Pandey and Olston use a collection of 48 weeks as opposed to our collection of 8 weeks. This allows them to use stochastic approaches such as a Poisson process to model change. Furthermore, their evaluation metric is closely tied into their schemes, and hence optimised for their schemes. Their implementations of various schemes make many assumptions in order to test them. For instance, the decision whether a document is relevant, the use of Zipf based click frequencies, and the simulation of results using a Poisson update distribution. Unlike our work, they do not incorporate the addition of new pages in any way. Finally, as noted earlier, this work was published two years after our work [Ali and Williams, 2003].

2.7.3 Data Warehousing

The issue of adaptively maintaining data consistency has been approached from a data warehousing perspective.

One study describes a scheme to maintain weak consistency of stock prices in a data warehouse by estimating a good Time-To-Live (TTL) value based on different measures [Srinivasan et al., 1998]. A TTL value indicates how long a data item is expected to match the source item and is typically used by proxy cache systems to predict the change frequency of cached items. In this work, Srinivasan et al. suggest several approaches for determining a "good" TTL.

- **A Static TTL Value**

  This scheme is easy to implement, however selecting a low TTL may result in an approach that contacts the web server too often, leading to wasted resources. While selecting a high TTL would save on network resources, this could lead to the data warehouse storing stale information.

- **A Semi-Static TTL Value**

  This scheme begins with a large TTL. It then reduces the TTL each time the source is found to be changing more frequently than it is being retrieved. The TTL can only be reduced, and so, the TTL will always be the worst case, even if the worst case occurs infrequently, leading to excessive polling.

- **A Dynamic TTL Value Based on the Most Recent Changes**

  This scheme assumes a low TTL initially leading to frequent polling. If the source changes frequently, the scheme polls frequently (low TTL). If the source changes infrequently, the scheme polls infrequently (high TTL). More recent changes are given greater emphasis, in this case utilising a history of two TTL values.

- **A Dynamic TTL Value with Static Bounds**

  To prevent the TTL value from becoming too high in cases where the source does not change for a long time, in this scheme a static bound is used to limit the maximum and minimum TTL value. This static value however, may not be representative of all sources.

- **An Adaptive TTL Value**

  This scheme uses a dynamic TTL with a dynamic upper bound that is based on the most rapid change observed so far.

Srinivasan et al. judge performance by how well a scheme minimises the metrics:

- **Number of Pollings**

  Measures the number of times the source is polled and indicates the network overhead of a scheme.

- **Violation Probability**

  Indicates the probability that a user's temporal consistency requirement is violated. This metric measures the duration that the difference between the local price and the source price exceeds the user constraint.

$$VProb = \frac{1}{T}\left(\sum_{i=1}^{n} t_i\right) \tag{2.27}$$

  where $t_1, t_2, \ldots, t_n$ denotes the durations during which the difference between the local price and source price exceeded the user constraint, and T is the total time that the data was presented to the user.

Using these metrics, they find that the adaptive TTL approach achieves the best results, producing the lowest probability of violation. The schemes are evaluated against real stock prices for several major IT companies over three-hour intervals. While these schemes appear to work well for rapidly changing stock prices, which have uniform format, it is unclear how well they would adapt to changing web documents.

In another study into data warehousing, Sundaresan et al. [2003] examine the problem of revisiting documents that have changed to keep a local data current. They examine the problem from the perspective of a distributed shared memory domain where a similar problem, memory coherency, exists. Being pull-based, in turn means that the data warehouse must poll the data sources (web documents) for changes.

Sundaresan et al. measure the average freshness of the warehouse collection and examine the ability of the data warehouse to effectively determine change frequency with varying levels of information availability.

- At level 1, only the updated data is available.
- At level 2, a timestamp — equivalent to the Last-Modified HTTP header — is available so that the age of the data can be determined.
- At level 3, a history of timestamps is available, making it possible to model the update rate based on a window of previous updates. The longer the history the less susceptible the model is to noise, however it will adapt to changes in update frequency much more slowly.

Sundaresan et al. find that using the last update (level 2) to estimate the next change produces the lowest freshness, but also has the lowest number of polls. Maintaining a list of all updates (level 3) produces significantly better freshness, but more than doubles the number of polls when compared to maintaining only the last update (level 2). Doubling the poll rate of the last update (level 2) scheme produces slightly better results than maintaining all updates (level 3) but requires nearly double the amount of polls required by the later scheme.

While these schemes work reasonably well, they rely on the accuracy of the timestamp information, something that cannot be done in a Web environment. Furthermore, they find that a simple push model, where each source informs the local warehouse, produced the best results, however this is obviously not feasible for search engines as outlined in Section 2.3.1.

They also consider different scheduling orders, such as *First In First Out* (FIFO), *Least Recently Requested* (LRR), and *Most Frequently Changed* (MFC). The LRR approach favours a model that does not over poll, and so, is more efficient, while the MFC scheme attempts to keep the most views up-to-date.

Sundaresan et al. find that, of all the schemes they compare, the MFC scheduling order produces the best overall freshness. The results are based on a Gaussian distribution of update intervals, and real trace statistics. While these results are interesting, they are not directly related to web document change, furthermore, change is implemented as a simple binary measure, and so, the implications of the result for real web documents is unclear.

In this section we have discussed various adaptive crawl ordering schemes that have been discussed in the literature. In the next section we discuss some of the past work into modelling the Web.

2.8 Web Change and Modelling

Many studies have examined the issue of web document dynamics and modelling portions of the Web. Web document dynamics are important for many areas of research, particularly crawling, data warehousing, and proxy caching.

2.8.1 Document Persistence

Often referred to as link-rot or web document half-life, document persistence is the study of the rate at which documents disappear from the Web. They use the half-life measure, loosely borrowed from physics, to determine the time taken for half of the documents to disappear.

A large study on document persistence by Koehler [2004] monitored 361 web documents on a weekly basis for approximately 6.5 years between December 1996 and May 2003. The results show that, although web documents are not particularly stable, once they have "aged" sufficiently, they tend to stabilise and are more persistent. Furthermore, resources from different domains, fields and disciplines tend to differ in their persistence.

In another study on document persistence, the authors examine the issue of link-rot of documents from biochemistry and molecular biology domains [Markwell and Brooks, 2003]. They monitored 515 science education resource web documents for a period of two years, and report a steady decline in the number of viable documents. They found that the "`.gov`" domain was the most stable, while the "`.edu`", "`.org`", and in particular, the "`.com`" domains were significantly more ephemeral.

Another study into the same problem in computing literature [Spinellis, 2003] examines the accessibility of URLs that have appeared in published material from 1995 to 1999 on the ACM digital library and the IEEE Computer Society digital library. The results show that 72% could be retrieved without problems. The remaining 28% could not be retrieved due to various issues regarding availability.

While these studies were concerned specifically about the disappearance of documents from the Web, others have examined the creation, modification, and deletion of documents.

2.8.2 Document Deletion, Creation, and Update

Several studies have examined the rate at which documents are created, updated, and deleted on the Web.

Pitkow and Pirolli use a survival analysis to examine the lifespan of documents on the Web. They consider the survival rate of documents that are mainly requested internally by the author, externally requested by others, and mutually requested by both the author and the external community. They find that documents that are mainly requested by external users have the highest survival rate, followed by documents that are mutually requested, and that internally requested documents have the lowest survival rate.

Lawrence and Giles [2000] showed that even the most powerful crawlers can take weeks or months to discover that a document has been created, deleted, or updated. This figure has substantially improved over the years.

Notess [2003] examined search results for six different queries and noted that most search engines had some results that were indexed in the past few days. Notess also shows that the bulk of the index was about one month old, and that some documents had not been re-indexed for much longer periods.

Lewandowski [2004] study date restricted queries to determine how effective search engines are at determining the correct date of a document. Using fifty random queries from the Fireball [2008] search engine, Lewandowski finds that the major search engines, Google [2008], Yahoo! [2008], and Teoma [2004], all have difficulty in this regard.

In another study, Lewandowski et al. [2006] compare Google [2008], Yahoo! [2008], and MSN [2008] finding that Google has the best overall freshness, with most documents being updated on a daily basis, however, only MSN maintains all documents with a freshness of less than twenty days. Recent studies of a large domain show that deep Web coverage has also improved [McCown et al., 2006].

Lim et al. [2001] crawl five popular sites to a maximum depth of five levels, twice a day, for one month and examine the extent to which documents change, and how "clustered" the changes are. They measure the degree of change by using word edit distance, and determine how clustered the changes are by dividing documents into blocks and determining which blocks contain changes. They devise two methods of dividing documents into blocks.

The first method divides the document into predefined blocks of 32 words, while the second method divides documents by paragraph "<p>" tags.

Their results show that 90% of documents have a change of less than 20%. Furthermore, most documents have changes that affect less than half the blocks when blocks are defined as 32-word groups. With paragraph tags are used to define blocks, they note unsurprisingly, that many documents have changes that affect less than half the blocks.

They conclude that, since changes are typically small and clustered, crawlers should use an incremental update approach to improve efficiency. That is, instead of discarding the collection and rebuilding the entire collection from scratch, only small subsets of the Web need to be recrawled to update the collection.

Ntoulas et al. [2004a] examine the evolution of web documents, monitoring 150 popular web sites for a year to determine the changes in both content and structure. Specifically their work examines changes that affect search engines, such as changes to link structure, the rate at which new documents and new content are created, the rate of change in content as measured by *tf.idf*, as well as the number of new words introduced.

Their study shows that new documents are created at an approximate rate of 8% per week, while 80% of documents are no longer available after one year. New documents tend to "borrow" content from existing documents. They estimate that 38% of content in new documents is "borrowed". They also show that the link structure of the Web is much more dynamic than the Web itself, with 25% of all links are new links being created each week and about 80% of links being replaced after a year.

Ntoulas et al. also show that documents tend to either remain unchained or go through small changes. There is less than 5% difference in 70% of changed documents after one week. About 50% of documents that are available after one year have no changes. Changes tend to be localised, with changes in restricted portions: weather, counter, reports, advertisements, and last update snippets.

Significantly, the results of Ntoulas et al. show that frequency of change is not a good predictor of degree of change, with no correlation between how frequently a document is changing, and how much the document changes. This effectively means that existing schemes that concentrate on frequency of change [Coffman et al., 1998; Cho and García-Molina, 2000c] would not effectively maximise the degree of changes detected. Their results also show that

past degree of change is a good predictor of future degree of change [Fetterly et al., 2003b]. That is, a document that changes by 10% in one week is likely to change by the same amount the next week.

Baeza-Yates and Castillo [2001]; Baeza-Yates et al. [2004] show that, unsurprisingly, new documents have poor PageRank, supporting our observation in Chapter 5 that finding documents that are both new and popular, is difficult. In our work in Chapter 5, we determine the popularity of a document by the frequency that it is returned in response to user queries, and we define a document as new if it has not been previous retrieved by the crawler.

A study by Koehler [2002] examines the stability, availability, and change rate of a set of documents over a four-year period. The study does not consider the creation of new documents over the time period, and so, represents the dynamics of an aging collection. In the work, a document is considered comatose if it cannot be successfully downloaded during six weekly requests. Koehler selected 361 URLs at random and retrieved them on a weekly basis during the period December 1996 and February 2001. The study examines the change in size of the documents, the number of new links, the changed items linked from the document, as well as the document purpose. Koehler uses two different definitions of document purpose, *navigational* and *content*. A navigational document is one whose main purpose is to direct users to the information the web site was designed to provide, contained within content documents [McDonnell et al., 2000].

The results show that document longevity is closely linked to domain type and document purpose, with commercial navigational documents having a better survival rate than content documents, while the opposite is true for educational documents. Koehler find that in general, navigation documents are more likely to survive than content documents. Furthermore, aging documents had a half-life of about two years, while the frequency and type of changes tend to become more stable over time. Aging documents were also less likely to be removed from the Web. Koehler argues that this may be due to the author becoming either satisfied with or disinterested in the document.

While these studies have examined web dynamics from a general perspective, others have also concentrated on web dynamics with respect to crawling.

	Top-Level Domain				
	All	Com	Edu	Gov	Netorg
Change < 1 day	23%	41%	2%	1%	11%
No change > 4 months	29%	14%	51%	54%	34%
Persistent > 1 month	72%	64%	84%	87%	71%
Persistent > 4 months	37%	26%	50%	56%	40%

Table 2.2: Change and persistence statistics of 720,000 popular pages. These values highlight relative volatility of the "`.com`*" domain, and the relative stability of the "*`.edu`*" and "*`.gov`*" domains*

2.8.3 Web Crawling and Web Change

The change rate of documents on the Web has been considered as part of a web crawling strategy to improve collection freshness. We first focus on several studies that have studied the link between web change and top level domains.

Top Level domains

Top level domains are the last part of an internet domain name, specifically, the letters after the final dot of a domain name. Many studies have highlighted the link between generic top level domain names, such as the commercial "`.com`" domain, and change frequency.

Cho and García-Molina [2000c] for example, monitor the daily evolution of 720,000 popular web documents over a four month period to determine how the Web evolves. They study how often web documents change, their lifespan, how long it takes for half the documents to change and model the change mathematically to determine how it affects crawling strategies. The results of their experiments, presented in Table 2.2, illustrate the volatility of the "`.com`" domain, and the relative stability of the "`.edu`" and "`.gov`" domains

They also find that 50% of all documents change or are replaced after approximately fifty days. The "`.com`" domain requires only eleven days for 50% of documents to change, while the "`.gov`" domain requires almost four months to have the same number of documents change. From this study, they conclude that web document change can be modelled according to a Poisson process and they use this in their crawling techniques.

Their work discusses web document change and how it affects various crawling strategies such as incremental and batch crawlers. Their definition of incremental crawling differs with respect to most of the literature. In their work they define an incremental crawler as one that continues crawling the Web once the crawlers resources are full and replaces less important documents in their collection with the newer documents. In contrast, they define batch crawlers as those that must periodically recrawl the Web and replace the entire collection with freshly crawled documents.

Their work does not consider in detail the problem of determining web resource change from document content. Rather, their crawler uses a simple checksum to determine whether documents have changed. It then examines the history of the changes to determine how many times the crawler detected that the document changed [Cho and García-Molina, 2000b; 2003a] and uses this in its scheduling strategy. Their results are then used to determine the features of an effective crawling technique. Specifically, they indicate that the average freshness of the collection is the same for both batch and incremental (steady) crawlers, if all documents are revisited every month by both schemes.

If both schemes retrieve the same number of documents over the same period of time, the batch crawler will have a higher peak speed. The batch crawler is not continuously crawling as the steady crawler does, and so, must collect documents at a higher speed during the times it operates. The paper also shows that updating the collection as the crawler is retrieving documents improves freshness but reduces availability. The paper describes an incremental crawler where document importance is measured using PageRank. Document importance is used to determine which documents to remove from the collection to make room for any new documents that are crawled. Occasionally the crawler decides to revisit documents to refresh the collection. In this case the crawler uses a checksum to determine whether documents have changed. It then examines the history of the changes, to determine how many times the crawler detected that the document changed [Cho and García-Molina, 2003a].

The paper does not consider how significant the changes are, simply whether they have changed or not. They state that importance could be used to determine if a document needs to be kept up-to-date. It is unclear whether the crawler they describe has been implemented. Furthermore, they do not consider the creation of new documents.

Douglis et al. [1997] analyse the following factors that affect how frequently resources change on the Web:

- Size of the resource.
- Resource type.
- Frequency of access.
- Resource location.
- Top-level domain

The work focuses on how these factors affect proxy caching. The research analyses requests and resources taken from two traces: one taken from a gateway containing the content of requests and response messages, and another taken from an Internet proxy.

Douglis et al. determine the fraction of requests that access resources that have changed, how old resources are when they are accessed, how modification times and access rates interact, how much duplication there is in the Web, and whether changes can be detected and exploited in HTML resources.

They find that, over a period of two weeks, many resources were modified to some extent, some were never modified, and a significant number were modified at least once between each trace. They also find that, although rate of change depends on many factors, content type was a significant factor along with top level domains, while the size of the resource had little correlation with rate of change. Their results are especially relevant to our work. In Chapter 4 we show that past change size is a relatively good predictor of change; however, we do not consider the size of the resource, but the change in size of the resource.

Fetterly et al. [2003b] also examine "top level domains", by monitoring 150,836,209 HTML documents on a weekly basis for eleven weeks to determine how frequently they change. They find that the amount of change varies dramatically for different top level domains, with documents in the "`.com`" domain changing more frequently than documents in the "`.gov`" and "`.edu`" domains. However, there is a much weaker relationship between top level domain and the amount of change. They also find that larger documents changed more frequently and significantly than smaller documents. Typically documents change in either

markup or in trivial ways. The relationship between the amount of change, the frequency of change, and document size is more significant for commercial domains than for educational and government domains. Finally they find that past change is a good predictor of future change and that the quality of documents is also important, considering the amount of spam documents they encounter, and the number of documents that change to a small degree in meaningful content.

Next we present studies that have examined web change without focusing on top level domains.

Other Studies

Wills and Mikhailov [1999a] discuss the accuracy of headers and their implications for caching. They use an MD5 hash to determine the accuracy of Last-Modified headers in test sets (home-pages taken from 100hot.com) over a two-week period.

As discussed in Section 2.1.1, they find that, in more than 9% of cases, resources had not changed, despite a change in the last-modified header. In approximately 0.3% of cases the resource had changed, despite no change in the last-modified header. They also show that in 14%–18% of cases no Last-Modified header was available.

They also performed a case study on several high-profile home-pages to determine short and long-term trends in changes, finding that many of the changes were predictable. That is, changes to the same few lines in the HTML code — particularly banner advertisements. This research heavily motivates our work in Chapter 4, that changes need to measured and analysed to determine their importance, particularly banner advertisements, and that the `Last-Modified` header is not a reliable indicator of change.

Brewington and Cybenko [2000a] present an analysis of web document changes and show that they do not follow a pure Poisson process as stated by Cho et al. Nevertheless, they assume that web document changes follow a Poisson process for their calculations.

In separate work, Brewington and Cybenko [2000b] discuss how fast the Web evolves based on daily observations of 100,000 documents over a period of seven months, and estimate the rate at which web search engines need to revisit the documents to keep them up-to-date. They model changes in web documents and use this to determine the frequency at which crawlers must re-index them to achieve particular levels of index consistency.

The age of a document is determined by calculating the difference between a document's Last-Modified header and the time it was retrieved. Their system monitors specific URLs for changes. In addition, the system also checks the top query responses whenever users present a query to the search engine, however a particular user's query will not be run more often than once every three days, unless, the same query is posted by another user.

Through their work, Brewington and Cybenko [2000b] find:

- A web crawler must download at least 45 million documents a day (a re-indexing period of 18 days) to maintain a 95% probability that a document taken at random is no more than one week old (assuming a Web of 800 million documents).
- A crawler must re-index at least 94 million documents a day (a re-indexing period of 8.5 days) to maintain a 95% probability that a document taken at random is no more than a day old.
- The Last-Modified header is available for around 65% of observations
- Most web documents are modified during the span of US working hours
- 4 kB–5 kB files with less than 2–3 images were modified most frequently.
- 4% of documents that were observed six times or more changed on every observation.
- 70% of these had no timestamp.
- 56% of documents observed six times or more had no change.
- 20% of the documents were younger than eleven days, with a median of roughly one hundred days.
- The older half of documents had a very long tail.

This research is closely related to ours, in that they too are examining the problem of determining frequency of document change. However, they are more concerned with modelling the change as opposed to determining the effect of the changes on query results. They do not consider the type, significance, and importance of the changes to documents in a search context. Their method of polling does not guarantee that all changes that occur

during any particular day are captured; however, they have conducted their experiments on a fairly large data set.

Whilst many studies have investigated web change, there are several common traits. Most use simple change metrics, despite studies showing that many modifications are trivial changes to content and markup, or predictable changes to banner advertising. Many assume change follows a Poisson process, despite observations to the contrary. Page content and purpose has a significant affect on change frequency, with commercial pages typically changing more frequently. Finally, many studies rely on past change statistics, the Last-Modified header, or some combination of the two, and do not consider the creation of new pages.

While these studies have examined web change and the implications for web crawling, other studies have examined the relationship between crawl ordering and its impact on the "quality" of the collection.

2.9 Crawl Ordering and Collection Quality

Crawl ordering techniques have typically been used as ways of improving the "quality" of the crawl collection. That is, given that the size of the Web is large and growing larger [Ntoulas et al., 2004a], crawlers need a way of ordering the crawl so that documents with the highest quality are crawled first.

In one study, Najork and Wiener [2001] investigate collection quality as 328 million documents are crawled in breadth-first fashion from the Web. They use the Mercator crawler [Heydon and Najork, 1999] and measure quality using PageRank [Page et al., 1998], finding that a breadth-first crawl performs well at the beginning of the crawl, downloading the highest quality documents. However, they find that the quality of the downloaded documents drops as the crawl continues. They believe that the reason for this is that high-quality documents have many inward links from different hosts, and so, are found early in the crawl regardless of where the crawl begins. The work does not compare breadth-first ordering to other crawling strategies.

A study by Cho et al. [1998] examines crawl ordering methods and their impact on collection importance as measured by PageRank [Page et al., 1998]. They compare ordering by backlink/Inlink count, PageRank score, and breadth-first, showing that breadth-first performs worst, while PageRank outperforms the other schemes. However, when combined with

query topic relevance, they find that results are mixed, with breadth-first outperforming other schemes for some topics but not for others.

In a study related to crawl ordering and collection quality, Boldi et al. [2004b] investigate partial graphs to determine how quickly the partial PageRank based ranking matches the complete PageRank ranking. They find that schemes that tend to miscalculate the final PageRank value also tend to gather high PageRank early.

Another study, by Baeza-Yates et al. [2005], compares different crawl ordering schemes to determine which retrieves "important" documents earlier in the crawl. They examine how document ordering strategies operate with different amounts of information available. Since a complete graph of the Web is only known at the end of the crawl, schemes cannot make an accurate and informed ranking during the crawl. However, if a crawl is being repeated, then there is historical information available with which to make a decision. They compare a strategy with all information to strategies with historical information and strategies with no extra information. Strategies with all information have access to a complete web graph oracle, which they can consult in order to select the document in the crawl frontier that has the highest PageRank score. Strategies with historical information can examine the web graph from a previous crawl to decide which document to retrieve next. Finally, strategies with no extra information have no access to information from past crawls, and so, can only inspect information that has been collected in the current crawl.

Strategies with no extra information:

- Breadth-First: Crawls documents in the order they are discovered.
- Backlink/Inlink count: Crawls documents with the most inlinks from documents already crawled.
- Batch-PageRank: Crawls documents with the highest PageRank score, recalculating PageRank each time K documents have been downloaded.
- Partial PageRank: Same as Batch-PageRank, however, between PageRank recalculations, new documents are assigned an approximate PageRank score using the sum of the PageRank of existing documents pointing to the new documents, divided by the number outlinks that the existing documents have.

- OPIC: On-line Page Importance Computation is a weighted inlink count, similar to a non-iterative PageRank score without any random links [Preda et al., 2003].
- Larger sites first: Crawls documents on sites with the most uncrawled documents, in an attempt to avoid having many unvisited documents from any one site.

Strategies with historical information use link graph data from the previous crawl to calculate PageRank for the current crawl. New documents are handled as follows:

- Historical PageRank Omniscient: Any new documents are assigned the PageRank from an oracle that knows the complete link graph.
- Historical PageRank Random: Any new documents are assigned a PageRank selected at random from the previous crawl.
- Historical PageRank Zero: Any new documents are assigned a PageRank of zero, and so, are crawled last.
- Historical PageRank Parent: Any new documents are assigned the PageRank of the documents that link to it divided by the number of outward links on the originating document.

They also consider a strategy with all information:

- Omniscient: Queries an oracle with knowledge of the complete graph. Crawls documents in the frontier with the highest PageRank.

Baeza-Yates et al. compare the different schemes by determining the PageRank of the collection as it is crawled, calculating the cumulative PageRank and the average PageRank.

Unsurprisingly, they find that the omniscient scheme is best. They find that backlink count and partial PageRank are the poorest, and are less effective than a random crawl. The breadth-first scheme outperforms most others during the beginning of the crawl, supporting the findings of Najork and Wiener [2001], but it drops in performance after crawling about 30% of the collection. Baeza-Yates et al. also find that the *batch PageRank*, *larger sites first*, and the *OPIC* schemes perform best, retrieving half of the available PageRank, while downloading 20%–30% of the documents. When compared to the *historic PageRank*

schemes, OPIC performs poorly to begin with, improving as link information is collected. The *historical PageRank* schemes outperform the *OPIC* scheme when comparing the average PageRank.

While in-depth studies have been carried out for measuring collection quality, few such studies have been conducted into collection freshness. Furthermore, while these crawl ordering schemes have been examined with regard to collection quality, many have not been evaluated in regard to collection freshness. Many of the studies into crawling that we have discussed in this section are directly related to our work. Similarly to our work, these studies examines crawl order, however, while they attempt to improve the collection quality with regard to importance, we attempt to improve collection freshness. Our work can compliment such crawl strategies to produce collections that are both fresh and of a high quality. In our work we examine many of these schemes in detail to determine how well they perform with regard to maintaining collection freshness.

In the next section we discuss other more general aspects of crawling that have been studied.

2.10 Parallel Crawlers

To improve the efficiency of crawlers and ensure that bandwidth is used effectively, it is important to parallelise their operation. There have been several studies examining issues related to parallel crawling.

In one study, Cho and García-Molina [2002] propose multiple parallel crawler architectures and discuss their characteristics. They define metrics for evaluating parallel crawlers and compare the architectures on a sample of 40 million documents collected from the Web. Their results suggest that in cases were less than five crawlers are running in parallel, a "firewall" approach is most effective. This approach segments the crawl space into predefined partitions that are assigned to each crawler, with overlapping URLs ignored by each crawler. In other cases, they suggest an approach that exchanges overlapping URLs with other crawlers.

In another study on parallel crawling, Shkapenyuk and Suel [2002] describe the design and implementation of a distributed, robust, and scalable crawler designed to process up to 400 documents per second. Finally, Boldi et al. [2004a] describe the implementation of UbiCrawler, a linearly scalable crawler.

In Appendix A, we discuss the implementation of our own LARA crawler, and the approach we use to parallelise its operation.

While general crawlers attempt to crawl all documents, topic-driven crawlers are a different kind of crawling application that attempt to retrieve the subset of documents that are related to a specific topic.

2.11 Topic-Driven Crawlers

Topic-driven or *focused* crawling attempts to gather only those documents that are relevant to a given topic Chakrabarti et al. [1999]. It orders the crawl frontier based on the relevance that URL anchor text has on a topic. Topic-driven crawling is based on the observation that documents sharing a link are more likely to be topically related [Davison, 2000].

Topic-driven crawling is particularly relevant to our work in Chapter 6, which uses a similar technique to topic-driven crawling. But, while topic-driven crawling measures the similarity between anchor text and a topic to order a crawl by relevance, we use anchor term statistics to measure the likelihood that anchor text point to a changed or new document.

One of the earliest examples of focused crawling was by Chakrabarti et al. [1999] and used example documents to search for other related documents.

Rennie and McCallum [1999] use reinforcement learning to improve the likelihood of crawling pages of a particular type or topic, producing a three-fold efficiency improvement over a breadth-first crawl.

Diligenti et al. [2000] applied context graphs to avoid pursuing short term gains in relevance, at the expense of following less relevant links that eventually lead to groups of highly relevant documents. They use search engines to train the crawler so that link distances between relevant documents can be determined for different topics.

McCallum et al. [2000] examine the use of machine learning to help in the construction of internet portals that are concentrated on a particular topic, through the use of topic-driven crawling.

Chau and Chen [2003] compare two traditional crawlers using breadth-first search, and PageRank, to a crawler using neural network techniques, to order a topic-driven crawl of a domain. In the case of the breadth-first search there is no way to ensure that documents were relevant to a topic. The PageRank scheme maintains two priority queues, one for URLs with

relevant anchor text and a second for all other URLs. The neural network spider maintains a graph of nodes that represent documents, and the links between the nodes representing anchor links. Each link is weighted according to the relevance of anchor terms to the topic, and links with a weight above a threshold are traversed, retrieving the documents they link to. The scheme then updates the graph, adding new nodes and links. The process stops once a predetermined number of documents are retrieved, or the average weight of nodes is lower than the allowable error. They develop a lexicon of both relevant, and irrelevant terms, which they use to compare anchor terms against in the case of the content-based schemes.

Chau and Chen also selected five high-quality hub documents as seed URLs. Relevant documents were determined by the percentage of relevant terms they contained.

Chau and Chen find that the neural network crawler performs best, with the breadth-first crawler performing marginally worse. Interestingly, they find that the PageRank crawler performed significantly worse than all other schemes, despite requiring the most computation time. The exceptional performance of the Breadth-First scheme was directly related to the choice of the high-quality seed URLs. PageRank performed exceptionally poorly since it placed great emphasis on documents with a large number of links that were typically irrelevant. Their work is of importance to us since it supports our results in Chapter 5 with regard to freshness.

Menczer et al. [2001] discuss methods of evaluating adaptive topic-driven web crawlers. They present three methods for evaluation: using classifiers trained on a sample set to assess newly crawled documents, ranking of crawled documents via an independent retrieval system and examining the order in which they were crawled, and finally, using cosine similarity to assess the mean similarity between each crawled document and the focused crawl topic.

Srinivasan et al. [2005]; Menczer et al. [2004] present an evaluation framework for topic-driven web crawlers. The system allows all logic related to the crawling algorithm to be encapsulated in a single module that is connected to the evaluation system via a standard interface. The system then keeps track of the resources used by the crawler, and the documents that are retrieved. Topics and relevant documents are retrieved from the DMOZ [2008]. The system measures the recall and precision of the crawlers in regard to a specific set of relevant target documents. The similarities of documents that are not in the target set are also evaluated to determine their degree of relevance with regard to the target set.

Chakrabarti et al. [2002] examine the information that is available about a document in the documents that link to it, and how this information can be used to accelerate topic-driven crawling. They implement a system that consists of two separate classifiers. The first of these is an *apprentice* that assigns priorities to unvisited URLs using features derived from the W3C Document Object Model [Hégaret et al., 2005]. The second classifier is a *trainer* that generates training instances for the apprentice. In effect, the trainer is a user specification of the desired content, while the apprentice learns how to find documents that match the desired content. Chakrabarti et al. show that their approach reduces the fraction of false positives by 30%–90%.

Next we discuss "Hidden Web" crawling, another specific type of crawler that are specifically created to retrieve documents that are only accessible via a search interface.

2.12 Crawling the "Hidden Web"

The aptly named "Hidden Web" is the collections of documents that are in databases and behind HTML forms. These documents are not directly linked by URLs, and so, crawlers cannot retrieve them during the normal crawling process. Instead, crawlers must retrieve them from within databases by completing HTML forms. This is typically achieved by posing queries to the database and examining the results. The queries can be retrieved from dictionaries, the general Web, from query results, or a combination of the above.

Raghavan and García-Molina [2001] introduce the Layout-based Information Extraction (LITE) technique, which automatically extracts semantic information from search forms and the results returned to queries.

Several studied by Ntoulas et al. [2004b; 2005] examine issues dealing with crawling the hidden web and compare methods that retrieve query terms from dictionaries, from the Web, and from returned query results. They show that an adaptive method that extracts terms from query results outperforms other schemes.

Callan and Connell [2001] present an effective query-based approach to sampling of databases on the Web, for the purpose of identifying the resource topics. This method, unlike the STARTS approach [Gravano et al., 1997] does not require cooperation of the resource providers, and is very similar to the approaches used for crawling the hidden web.

Lin and Chen [2002] discuss a system for automatically searching the hidden web. The system is essentially a meta-search engine that maintains details of the various hidden collections and automatically selects the relevant collection to query when a user poses a query.

While most work has concentrated on extracting information from the databases, Wu et al. [2004] focus on accurately matching the interface of search interfaces via an interactive clustering-based approach.

These studies highlight the many difficulties that arise when crawling the Web. In our work we do not attempt to crawl pages hidden behind forms. It is therefore unclear how well our schemes would operate in such environments.

Next we discuss the methods employed by crawlers to detect duplicate collections on the Web, known as mirrors.

2.13 Mirroring

The vast disparity in available bandwidth across the Web, and the multiple web users frequently requiring the same data at the same time, leads to an excessive strain on web servers. To alleviate this problem and improve load sharing, web administrators can use mirroring techniques. Mirroring replicates groups of documents or entire web sites at multiple URLs. While mirroring is beneficial to users, it can be a problem for web crawlers. If crawlers retrieve multiple copies of the same data, this is not beneficial to search engine users, since repeated search results do not provide new information. Furthermore, recrawling the same data at a different location wastes crawler bandwidth that could otherwise be used to improve search results. Crawlers therefore need to be able to detect mirrors and avoid crawling them unnecessarily. Propagation delay can further complicate detection of mirrors, since mirrors may not be identical.

A study by Fetterly et al. [2003a] examines the number and distribution of "near-duplicate" documents on the Web. They measure similarity using five word shingles [Broder et al., 1997] and define "near-duplicate" documents as documents that share two "supershingles". They collected data on a weekly basis over a duration of eleven weeks, consisting of approximately 150 million web documents. Their results show that 29.2% of these documents are very similar, while 22.2% are virtually identical. Many of these near duplicates remain so over time, with implications for mirroring.

In other work, Cho et al. [2000] identify techniques for detecting mirroring and examine 25 million documents for mirroring. They use a content similarity measure that determines textual overlap [Shivakumar and García-Molina, 1995], and consider both document similarity and link similarity to identify mirrors.

In a large study, Bharat and Broder [1999] examine 179 million URLs, and 238,000 hosts to determine how much mirroring there was on the Web. They define two hosts as mirrors if a large percentage of URL paths are valid on both domains, and the common paths contain documents with similar content. Specifically, they examine structural and content similarity. Structural similarity is defined by the relative paths on a host. When two different hosts have the same set of paths, they are classified as *structurally identical*. On the other hand, if two documents are byte-wise identical they are classified as *content identical*. If documents have undergone changes at the byte level, such as the addition of white-space or HTML reformatting without changes to content, they are no longer content identical. Instead, two documents are considered *content equivalent* if they are identical after they are normalised for such changes. If documents undergo changes due to banner advertising or other dynamic content, but remain highly similar at a syntactic level, they are considered *highly similar*. Similarity at a syntactic level is measured using shingling [Broder et al., 1997]. Finally, two documents are considered *related* if they change substantially at the syntactic level but are semantically similar. Using these measures of similarity they define six levels of mirroring:

- **Level 1: Structural and Content Identity.**
  - *Every* document on Host A is replicated with *byte-wise identical* content on Host B and vice versa.
- **Level 2: Structural identity. Content equivalence.**
  - *Every* document on Host A is replicated with *equivalent* content on Host B and vice versa.
- **Level 3: Structural identity. Content similarity.**
  - *Every* document on Host A is replicated with *highly similar* content on Host B and vice versa.

- **Level 4: Partial structural match. Content similarity.**
  - *Some* document on Host A is replicated with *highly similar* content on Host B and vice versa.
- **Level 5: Structural identity. Related content.**
  - *Every* document on Host A is replicated with related but not syntactically similar content on Host B and vice versa.
- **Mismatch: None of the above.**

They also investigate whether the IP address of hosts are related. They compare paths by tokenising them by directory and creating word bi-grams on consecutive directories. They convert characters to lowercase, treat non-alphabetical characters as word breaks, eliminate stop words, and ignore features that occur only once in a host. Finally, they compare document content with the use of shingling [Broder et al., 1997]. In their experiments they find that 10% of the hosts in the collection are mirrored.

In Appendix A, we describe the design and development of our Lara crawler, however we avoid issues related to mirroring by manually avoiding them. If we were to conduct a general crawl of the Web, we would need to implement some, or all or the techniques we have described.

While we have discussed maintaining freshness and consistency of collections retrieved by web crawlers, a similar problem is encountered with proxy caching.

2.14 Proxy Caching

Proxy caches operate in multi-user environments and attempt to reduce web traffic by storing local copies of web resources that are frequently requested by local users. The are two main issues regarding proxy caching:

- **Cache replacement policies**

  There are potentially unbounded numbers of resources on the Web, due to documents that are dynamically generated [Baeza-Yates and Castillo, 2004]. For instance, online calendars, diaries, and sites that calculate the value of π, can dynamically generate an

unbounded number of documents. This coupled with limited local storage capacity, dictate that proxies need to decide on which resource to replace in the local cache once it is full. This issue is generally not directly relevant to crawling.

- **Cache consistency**

  Proxy caches store local copies of web resources, and so, as with crawling, need to deal with issues regarding synchronising the local copy of a resource with the source.

Chankhunthod et al. [1996] discuss a proxy cache system that has different levels of hierarchical caching with sibling and parent cache to provide greater flexibility and scalability.

Belloum and Hertzberger [2002] examine the impact of dynamic documents on a cache replacement policy. They show that the performance can be improved dramatically by prefetching cached documents when they become stale according to their TTL value, before they are retrieved by users. In this sense it operates much like a crawler for a search engine, since updates are not in response to user requests.

Cao and Liu [1998] analyse three different approaches to maintaining strong cache consistency in a proxy cache system. They compare an adaptive TTL, a poll every time, and an invalidation approach. The adaptive TTL approach that they use is the same as that proposed by Cate [1992]. The poll every time approach sends an `if-modified-since` request whenever a user requests a resource contained in the cache. The `if-modified-since` request informs the web server to return the requested resource only if it has been modified since the date that the client last retrieved the resource, which is provided by the client as part of the request. Finally, the invalidation approach relies on the server to notify clients when files are modified.

Cao and Liu show that an invalidation approach generates a similar volume of network traffic and server workload as an adaptive TTL approach, yet maintains consistency more effectively. In addition, the invalidation approach maintains consistency as effectively as the poll every time approach, but produces a much smaller volume of network traffic and server workload. The invalidation approach, however, is not currently supported by the HTTP protocol, and may be difficult to implement on the Web, making it unsuitable for web crawling.

An alternate approach towards improving collection consistency has been to improve crawler and web server cooperation, which we discuss next.

2.15 Crawler/Server Cooperation

Several studies have examined ways of improving web servers to better accommodate crawlers and improve web crawling efficiency.

McLearn [2002] examine how cooperation and information sharing between competing crawlers can improve collection freshness, increase web coverage, reduce redundancy, and reduce resource usage. They define freshness as a crawlers ability to capture "web events", such as the addition of new web objects or changes made to them. They formalise their definition of freshness during interval t_i and t_j as:

$$\text{Freshness } t_{ij} = \frac{\text{Number of captured web events } t_{ij}}{\text{Total number of web events } t_{ij}} \tag{2.28}$$

Brandman et al. [2000] investigate methods for improving web servers to better accommodate web crawlers. They propose that web servers export meta-data describing content and last modified information. In the absence of these kinds of improvements or a push-based models, the current method of periodic polling is required to keep collections fresh.

Gupta and Campbell [2001] propose a method that allows web servers to monitor changes in their local files and push the changes in batches to search engines. Specifically they incorporate popularity, age, and freshness to decide which documents changes to push to search engines, and measure the number of bytes transmitted to determine the transmission costs. The algorithm is analysed but not implemented, and so, its effectiveness is unclear. Furthermore, this approach requires cooperation from web servers to accurately keep track of changes to hosted documents.

Mikhailov and Wills [2002; 2003] propose the MONARCH approach for strong cache consistency, which runs on the web server and monitors objects for change, assigning cache invalidation commands accordingly. The approach assumes that documents consist of multiple objects that must be retrieved to render a document, and that these different object can change at different rates. Caches are then instructed to request or validate the most frequently changing objects on every request. The requests are also used to determine which

other objects the client may have cached. The relationships between objects in a document are used to determine which object in a document is the most dynamic and therefore is used to "manage" the other objects in the document. This reduces the number of objects that the server must monitor for changes. While this reduces server workload, it still requires wide adoption to impact search engine performance and policy.

There have also been submissions by private groups to the W3C proposing standards for improving the efficiency of replication on the Web [W3C, 1997]. They propose standards for maintaining an index of files so that proxy clients can quickly determine which files have changed and download them to ensure the consistency of changed files.

Other groups have also proposed methods for efficiently searching and locating resources from web servers [DASL, 2000].

While many approaches have been proposed to improve web server and search engine cooperation in an effort to improve search engine index freshness, none have been adopted in the general Web, and so, search engine still rely upon heuristic methods to predict document change frequency.

Next we discuss the various crawler designs and implementations that have been published in the past.

2.16 Crawler Implementation

There have been many papers published in the past describing the design and implementation of web crawlers. In Appendix A we describe the implementation issues with our Lara crawler, which we used to gather documents for our work. Here we highlight the design and development of published crawlers in chronological order.

Some of the first published crawlers include the RBSE spider [Eichmann, 1994], WebCrawler [Pinkerton, 1994], and the World Wide Web Worm [McBryan, 1994]. The Archive crawler, another early crawler implementation, has been continuously crawling the Web since October 1996 and allows users to examine changing snapshots of the Web over that time [Burner, 1997; Internet Archive, 2006].

Other authors report on toolkits for designing crawlers. Miller and Bharat [1998] for example, describe SPHINX, a Java toolkit that provides a framework for designing and implementing a web crawler.

Several crawlers used by large commercial search engines have also been described in past studies. An early implementation of the famous Google [2008] crawler and search engine is described by Brin and Page [1998], while Heydon and Najork [1999] describe the design and implementation of Mercator, a scalable and extensible crawler, used by the Altavista [2008] search engine.

Several studies have described the design of crawlers that incorporate techniques for maintaining the freshness of crawled documents. For instance, CoBWeb, a scalable and distributed web crawler for the Brazilian Web is described by da Silva et al. [1999]. The design of CoBWeb considers issues related to updating previously crawled documents by utilising a cache parasite approach. This obtains information from proxy cache servers about URLs that have been requested by users. Then any new or updated documents are requested directly from the proxy cache. A study by Edwards et al. [2001], describes the design of the WebFountain crawler, and the optimisation model used to control the crawl strategy to specifically improve the freshness of the collection. While, Baeza-Yates and Castillo [2002] describe the implementation of a crawler that considers collection freshness and other important factors such as document importance.

Several past studies have examined the development of highly distributed crawlers for the purposes of parallelisation. Shkapenyuk and Suel [2002] for example, describe the design and implementation of a distributed web crawler theoretically capable of retrieving up to 400 documents per second, though limited to 300 documents per second due to network constraints. Shkapenyuk and Suel [2002] describe the design and implementation of a distributed, robust, and scalable crawler designed to process up to 400 documents per second. Zeinalipour-Yazti and Dikaiakos [2002] present WebRace, a distributed web crawler, filtering processor and object cache that is incorporated into Internet middleware infrastructure. Risvik and Michelsen [2002] give an overview of the FAST crawler, capable of retrieving 400 documents per second that was used to serve the Alltheweb [2008] search engine. On a related issue, Broder et al. [2003] examine methods for caching the "seen URL" list for large scale crawlers.

Other papers describing crawler implementation include work by Boldi et al. [2004a], Hafri and Djeraba [2004], and Shokouhi and Chubak [2004].

Next we discuss other various crawler related topics that have been covered in the literature.

2.17 Other Crawler Related Studies

Lawrence and Giles [1998] examine the coverage of search engines finding that no single search engine indexed more than one-third of the "indexable" Web.

In another study Gulli and Signorini [2005] show that the indexable Web is more than 11.5 billion documents as of January 2005 and that even Google [2008], the largest search engine at the time indexed only 76.2% of these documents. Furthermore, the combined coverage of the four major search engine — Google [2008], Yahoo! [2008], Ask.com [2008], and MSN [2008] — was only 9.36 billion or 81.4% of the indexable Web at the time.

Czumaj et al. [2001] examines the issue of access times for documents across the Web. They show that access times fluctuate according to time zone, and that this can be used to schedule crawling during less busy times.

Castillo [2004] begins by describing a web crawler model that integrates the crawler with the rest of the search engine. They present methods for retrieving important documents early in a crawl [Baeza-Yates et al., 2005], where importance is measured as PageRank. They also model user behaviour to determine how crawlers should traverse web sites, and propose web server cooperation methods to improve search results and reduce network congestion.

They model a random surfer on an infinite Web to determine how deep a user would go into a web site, where depth is measured as link depth. At each point the surfer can:

- go to the *next* level
- stay at the *same* level
- go to the *previous* level
- go to a *different deeper* level
- go to a *different previous* level
- go to the *start* document
- *leave* the site

From this model and web logs, they show that a crawler would not have to traverse deeper than about four links, in the case of web sites, and seven links, in the case of web logs or

"Blogs", to capture most of the documents that users would visit. They also discuss the implementation of the WIRE crawler, which considers both freshness and quality [Baeza-Yates and Castillo, 2002].

2.18 Summary

In this chapter we have highlighted the various difficulties associated with crawling. While there has been work on many aspects of crawling, very little work has examined the problem of maintaining freshness of crawled documents. Furthermore, many of the studies on this aspect have made assumptions about the rate of change of documents, have used simple notions of change, or have simulated their experiments. Studies that have examined change detection have not examined the impact these changes have on search results. Furthermore, while many studies have been made into the effect of crawl ordering on collection "quality", few have examined their affect on collection freshness. While several studies have discussed the use of user feedback in crawl ordering from a theoretical standpoint, few have examined this approach empirically. Finally, adaptive schemes have typically relied on a long history of change to predict future change, or have applied a single crawl strategy for an entire site. In our work we first set out to determine an effective meaningful change metric that is computationally inexpensive. We define a novel approach to measuring change that can identify changes that will have a real impact on user search results. We then produce test collections for our experiments, such that real world performance can be measured. Next, we test some traditional and novel stateless crawl ordering schemes to determine their affect on collection freshness. Finally, we test our novel adaptive crawling scheme, which does not require a history of changes and effectively orders a crawl such that documents that have changed are more likely to be recrawled.

Chapter 3

Experimental Environment

As we described earlier, web search engines store collections locally in order to provide fast, scalable search facilities. The collections are gathered by crawling the Web. A problem with crawling is determining when to revisit resources that have changed: stale documents contribute towards poor search results, while unnecessary refreshing costs search companies both time and money.

Evaluating crawler performance with regard to collection freshness requires several key components: a test collection, a measure of resource change, and an evaluation methodology. Together, these form an evaluation framework or system, something we discuss further in the next chapter. In this chapter, we describe the first of these components — our test collection and how it was created.

3.1 Introduction

Crawlers operate in an environment where autonomous changes are made to target documents. To maintain consistency of collections, crawlers must periodically revisit documents after they have been initially crawled. Evaluating their performance in this regard is difficult. The volatility of the Web makes it impossible to consistently evaluate crawling schemes directly. Instead, either web resource change must be modelled, or test collections must be created. As outlined in Section 2.6 (page 55), previous work has proposed that changes on the Web can be modelled according to a Poisson process, however some have shown that changes on the Web are not according to a pure Poisson model [Dasen and Wilde, 2001], and

other large studies have shown that in fact the Web does not change randomly [Brewington and Cybenko, 2000a; Padmanabhan and Qiu, 2000]. In our work, rather than attempting to model the Web, or make assumptions about the way in which it changes, we use snapshots of the Web and test our schemes against these test collections, a process that we describe further in the next section. In addition, we describe our motivation for this test collection, and how it was developed.

3.2 Test Collection

While crawlers are normally used to retrieve documents from live, dynamic web collections, they cannot be tested directly against live collections. Since various crawling schemes need to be compared, and results need to be repeatable, it is necessary to use a static web. Testing a scheme directly against the Web would most likely produce different results each time, making it impossible to compare the results of any schemes. To avoid these problems, test collections or change modelling techniques are required. In past work, different models have been used to describe change on the Web, however many have used simple notions of change, making it difficult to model meaningful change on the Web. Instead, in our work we have opted to use a test collection as this would give us statistics representative of the real Web.

Many typical IR tasks have test datasets, such as the TREC [Voorhees and Harman, 2005] text retrieval collections, the INEX [Fuhr et al., 2004] XML data retrieval collections, and the TRECVID [Kraaij et al., 2006] video retrieval collections, with which to run their experiments. However, there are no collections for testing crawler index consistency algorithms. Furthermore, in order to evaluate the performance of a crawler, statistics about the rate of change of all documents are required, not just the documents that are recrawled. These two problems necessitate the use of a testbed that contains a copy of each available document, sampled at a regular interval.

Our collections are produced from two large Australian web sites. The first of these is the Australian Broadcasting Commission (ABC) web site. This is a large domain consisting of about 500,000 documents, of about six gigabytes in total. We crawled the ABC on eight occasions in one-week intervals.

Crawl	Documents	Size (MB)	Date
1	485,190	6,522	27 May 2004
2	465,684	6,440	04 June 2004
3	475,451	6,481	11 June 2004
4	563,670	6,799	18 June 2004
5	611,819	7,751	27 June 2004
6	614,393	7,833	05 July 2004
7	546,630	6,850	22 July 2004
8	540,530	6,871	30 July 2004

Table 3.1: Statistics of the documents retrieved during each of the eight crawls of the ABC.

Interval	New	Removed	Changed	Unchanged	Changed Popular	New Popular
1-2	10,197	29,703	44,049	411,438	19,994	4,461
2-3	31,190	21,423	40,878	403,383	20,667	9,055
3-4	96,767	8,548	78,032	388,871	39,015	33,111
4-5	71,910	23,761	45,696	494,213	17,308	29,208
5-6	17,654	15,080	64,126	532,613	22,417	4,962
6-7	19,772	87,535	49,323	477,535	21,354	7,222
7-8	10,935	17,035	54,181	475,414	22,363	3,923

Table 3.2: Number of documents that have changed, not changed, been added, or removed from week to week in the ABC collection. The last two columns are statistics of popular documents, determined by query statistics. Change is evaluated using Words $\alpha = 10$.

Crawl	Documents	Size (MB)	Date
1	143,143	1,171	05 Oct 2004
2	141,905	1,064	13 Oct 2004
3	134,836	1,047	19 Oct 2004
4	144,499	1,209	25 Oct 2004
5	147,134	1,157	29 Oct 2004
6	145,372	1,146	05 Nov 2004
7	149,137	1,159	10 Nov 2004
8	118,555	1,003	15 Nov 2004
9	137,116	995	19 Nov 2004
10	151,791	1,162	25 Nov 2004
11	95,343	738	01 Dec 2004

Table 3.3: Statistics of the documents retrieved during each of the eleven crawls of CSIRO.

Interval	New	Removed	Changed	Unchanged	Changed Popular	New Popular
1-2	10,165	11,403	3,502	128,238	1,095	2,172
2-3	10,968	18,037	3,993	119,875	1,475	4,066
3-4	15,348	5,685	5,141	124,010	1,588	4,152
4-5	11,415	8,780	5,156	130,563	1,775	3,456
5-6	9,665	11,427	4,624	131,083	2,096	3,793
6-7	9,775	6,010	7,295	132,067	2,973	4,161
7-8	11,533	42,115	4,081	102,941	1,568	4,148
8-9	38,283	19,722	3,226	95,607	1,285	9,239
9-10	20,539	5,864	5,305	125,947	2,709	5,852
10-11	6,370	62,818	2,600	86,373	1,277	2,148

Table 3.4: Number of documents that have changed, not changed, been added, or removed from week to week in the CSIRO collection. The last two columns are statistics of popular documents, determined by query statistics. Change is evaluated using Words $\alpha = 10$.

The second collection was taken from the CSIRO, a site which is representative of a typical large organisation, due to the large percentage of well connected internal links [Broder et al., 2000a;b; Fagin et al., 2003a]. This collection consists of about 120,000 documents, of approximately six gigabytes in total, which we crawled on eleven occasions, also in one-week intervals.

The decision to conduct our study using the ABC and CSIRO web sites was based on their large sizes and the substantial variety of documents that they contain, ranging from television and radio program information, to news articles, transcripts of audio, information about staff members, and web forums. Access to the query logs for each site also allowed us to analyse the utility of a popularity metric, and a query-biased crawling scheme, discussed in Chapters 5 and 6.

Statistics from the crawls are presented in Tables 3.1 through to 3.4. In Tables 3.1 and 3.3 we present the number of documents retrieved during each crawl, the size of the crawl and the date it was conducted. In Tables 3.2 and 3.4 we show the number of documents that moved in and out of the collection on a week-by-week basis. Also shown in the final columns of the each table is the number of the changed documents that were returned in the top thousand documents to any query in the period (taken from query logs): that is the documents that have changed and are deemed popular. We note that there is a mismatch in the time when the queries for the CSIRO were posed and the time the CSIRO collection was crawled, which could lead to the wrong documents being flagged as popular. Specifically, the queries for the ABC collection were from the time period 27th May 2004 to 30th July 2004, while the queries for the CSIRO were from the time period 19th June 2004 to 19th August 2004. We refer to these tables more in Section 5.4.

Both crawls were in breadth-first order, beginning with the root document for each domain; http://www.abc.net.au/ for the ABC domain and http://www.csiro.au/ for the CSIRO domain. Crawling was with our own Lara crawler, the design and development of which we discuss in Appendix A. The crawls were restricted to only those documents that were accessible from internal links within the sites. This in turn means that we have no information external to these two domains, including the link graph data that is used for computing PageRank, inlink, and hub statistics for schemes discussed in Section 2.2.2.

Both domains were crawled with approximately one-week intervals, though there is a two-week gap between crawl six and seven of the ABC. This crawl rate imposes restrictions on our results: our testbed does not allow us to accurately model changes that occur more frequently than approximately once a week or less frequently than approximately once every eight weeks for the ABC and once every eleven weeks for the CSIRO. While these collections are not used to examine our schemes in Chapter 4, they are used extensively in our experiments in Chapters 5 and 6. Our work in the Chapter 4 does not require ongoing statistics over multiple crawls, nor the URL link structure that is required for our experiments in later chapters. Furthermore, our experiments in Chapter 4 require documents that are collected from heterogeneous domains so that we can produce generic change metrics. Nonetheless, we do test our change metrics in Chapter 4 against the ABC collection to ensure that they are suitable in later chapters.

3.3 Summary

Web crawling is essential to the maintenance of an up-to-date and complete document collection for a web search engine. Testing crawling algorithms directly against the Web does not produce consistent results due to document variations over time, and so test collections are necessary. In this chapter we have highlighted the test collections which we use to evaluate our algorithms in later chapters. As we have noted, test collections are an important component of a crawler evaluation framework. In the next chapter we outline the two other components — a measure of resource change, and an evaluation methodology.

Chapter 4

Crawler Evaluation Framework

Web crawlers collect resources from the Web so that they can be stored locally and indexed by search engines. Crawlers must also periodically revisit these resources to maintain the local collection.

As we highlighted in the previous chapter, evaluating crawler performance with regard to collection freshness requires several key components: a test collection, a measure of resource change, and an evaluation methodology. Together, these form an evaluation framework or system. In the previous chapter we discussed our test collection. Here we investigate the two other components — the measure of change, and the evaluation methodology.

While many studies have examined measures of resource change, relatively few have examined them from a crawler perspective and even fewer have examined their impact on search engine user experience. Measuring meaningful change is important since some changes — such as in images, advertisements, or headers — are unlikely to affect query results.

In this chapter, we investigate measures for determining whether documents have changed and should be recrawled. We highlight our crawler evaluation framework, consisting of our approach to measuring meaningful change, and our approach to evaluating the impact of change on user experience.

We empirically test our change metric and show that it is more effective than other existing simple measures at identifying meaningful change. We also show that content-based measures are more effective than the traditional approach of using HTTP headers for detecting meaningful change, that is, change that affects users' search results.

Refreshing based on HTTP headers typically recrawls 15% of the collection each day, but users do not search for the majority of refreshed documents. In contrast, refreshing documents when more than ten words change recrawls 22% of the collection, but updates documents more effectively, detecting more of the changes that affect users' search results. We conclude that our approaches are an effective component of a web crawling strategy.

4.1 Introduction

Crawlers operate in an environment where autonomous changes are made to target documents. To maintain consistency of collections, crawlers must periodically revisit documents after they have been initially crawled. Evaluating their performance in this regard is difficult. The volatility of the Web makes it impossible to consistently evaluate crawling schemes directly. Instead, either web resource change must be modelled, or test collections must be created. As outlined in Section 2.6 (page 55), previous work has proposed that changes on the Web be modelled according to a Poisson process, however some have shown that changes on the Web are not a pure Poisson model [Dasen and Wilde, 2001], and other large studies have shown that in fact the Web does not change randomly [Brewington and Cybenko, 2000a; Padmanabhan and Qiu, 2000]. In our work, rather than attempting to model the Web, or make assumptions about the way in which it changes, we use snapshots of the Web and test our schemes directly against these test collections, a process that we described earlier in Chapter 3 (page 95).

Another drawback with existing crawler evaluation techniques is that they typically do not consider search engine user experience. While we outlined a wide range of change detection metrics in Section 2.5.1 (page 51), many of these require large amounts of computation in addition to standard search engine functions, and few have been analysed to assess their impact on crawling performance and search results that is, their effectiveness from a crawling perspective, something we investigate further in this chapter.

Our work in this chapter also compares several change metric approaches against the more traditional approach of using HTTP headers. Inspecting the expiry and last modified information in the HTTP headers of web resources is a well-known technique for establishing when to recrawl a resource [Srinivasan et al., 1998; Brewington and Cybenko, 2000b; Wessels, 2001a]. However, as we discuss further in Section 4.5.1, this information is only present

in around 39% of the documents in our general web collection (PROXY) — a results that is in marked contrast to the observations of others (82%–86% and 65%) as discussed in Sections 2.1.1 (page 25) and 2.8.3 (page 73).

Furthermore, while web crawlers can request just the HTTP header information and use this to determine whether a recrawl is necessary, this does not guarantee improved efficiency or effectiveness. While this may lead to reduced bandwidth usage since the entire document is not retrieved, the process nevertheless involves using a HTTP request which could have been used to retrieve a page. Furthermore, as we have noted previously, there are reliability and availability issues with the use of HTTP headers to predict resource change.

Another approach is to compute the past change in a resource and use this to decide whether a document has changed enough to warrant recrawling. We explore this idea in this chapter by investigating whether the number of bytes added or removed from a document is a good indicator of interesting change and, if so, how many bytes of change is significant enough to impact on search results. We extend the idea for words, and also investigate the impact of removing HTTP headers, document markup, and common words prior to computing change.

Our results show that our metrics are excellent predictors and detectors of changes that have an impact on search results. For example, our WORDS metric can predict about 93% of changes affecting rank while recrawling just 22% of the collection. In contrast, the SHINGLING change metric, typically used by others, must retrieve 3.5 times as many pages.

Our results also demonstrate the limitations of using HTTP headers to predict change. The HEADERS approach to predicting change typically detected 10%–15% of rank variations due to stale documents, but recrawled about 14% of the collections. Our WORDS content based change prediction metric detected the same amount of variations while retrieving 2%–4% of the collection.

We conclude that our simple change metrics are an effective component of a web crawling refresh strategy. In addition, we recommend our accuracy evaluation approach as a benchmarking method for determining crawler accuracy.

In the next section we outline some simple change metrics that can be implemented into the crawling and indexing process. We then follow with a proposal for a novel approach to evaluating their performance that directly measures their impact on search results.

Finally, we use this evaluation technique to identify an effective, meaningful change detection tool that we use as part of our experimental framework in later chapters.

4.2 Document Change Metrics

In this section, we propose simple measures for detecting changes and determining the degree to which these changes impact upon a user's search results. We examine measures that are efficient to compute: some of our measures can be calculated during the normal process of indexing; that is, they require no additional document processing. In our experiments we examine how well each metric detects meaningful changes, as well as their capacity to predict future change. We examine the performance of each metric using intervals, specifically time t, $t+n$ and $t+2n$.

We propose four metrics that we test in Section 4.5.

1. NONE— retrieve all documents at time $t+n$ but none of the documents at time $t+2n$. In this way, all other metrics are guaranteed to perform as well as the NONE metric, since they will contain all documents from time $t+n$ and some subset will be replaced with updated documents from time $t+2n$ as the threshold for change α is reduced.

2. ALL— retrieve all documents at time $t+2n$. Hence every document is up-to-date.

3. SIZE— retrieve all documents at time $t+n$ and the subset of documents from time $t+2n$ that have changed by more than α bytes in file size between time intervals $t+n$ and $t+2n$. Importantly, our basic implementation of SIZE is based on the raw file size of the document and does not include any form of document preprocessing such as removal of markup or stop words. The SIZE metric is extremely efficient to calculate since it based on the file size of a document.

4. WORDS— retrieve all documents at time $t+n$ and the subset of documents from time $t+2n$ that have changed by more than α words between time intervals $t+n$ and $t+2n$. A word is defined as a string of alphanumeric characters that may include hyphens or apostrophes.

As discussed in Section 4.1, we experiment with the use of the HTTP headers `Expires` and `Last-Modified` (which we refer to as HEADERS). We also experiment with the *shingling*

method discussed in Section 2.2, with no recrawling which we refer to as NONE, and with full recrawling which we refer to as ALL.

We use the ALL and NONE metrics as ceiling and floor benchmarks for performance respectively. The ALL metric shows the maximum achievable accuracy for any of the metrics that we test, and is produced when all documents are updated in the final interval $t+2n$. All other metrics that we test will always retrieve either the same set of documents or a subset thereof. Because of this, ALL crawls the maximum number of resources per time interval. In contrast, NONE demonstrates the impact on search accuracy when a crawler fails to update a collection. We do not propose using ALL or NONE in practice.

The SIZE metric is an approximation of ALL but any changes that do not affect size by more than α bytes are ignored. Moreover, even when $\alpha = 0$, document changes can occur that are not detected: for example, a changing date, page number, or HTML colour attribute may not affect the size of the document. Because it is less sensitive to change, we expect that, for low values of α, the SIZE metric will be effective. In addition, because SIZE can be computed from file size, it is very efficient to evaluate.

The WORDS metric is sensitive to single byte changes, but not to more than one change per word. In this context, change is defined as substitution, deletion, or insertion of a word, that is, α defines the number of such elementary changes that must occur for a document to be crawled. This is a different form of robustness to change to that of the SIZE metric: a multi-byte change in a currency value, a counter, or a document footer may be only a one-word change. The WORDS metric therefore matches an intuitive understanding of what constitutes change. Words are defined as strings of alphanumeric characters that may include hyphens or apostrophes; all other characters are non-words and define boundaries between adjacent words [Williams and Zobel, 2005]. We illustrate the WORDS metric with an example in Figure 4.1. Note that if we use the SIZE metric to measure the change we get a difference of three.

The *shingling* method, discussed in Section 2.2 (page 52), measures the number of sequences of length n that two documents have in common. In the experimentation reported in this chapter, we measure *resemblance* using a length of seven words, as used in work by Edwards et al. [2001].

(A) Lara Croft is a tomb raider

(B) Brian Lara is a cricket player

	Lara	Croft	is	a	tomb	raider
↓I		↓D			↓S	↓S
Brian	Lara		is	a	cricket	player

Figure 4.1: Measuring the difference between sentence (A) and (B) using the WORDS *metric. In this example an* S *indicates a Substitution, a* D *indicates a Deletion, and an* I *indicates an Insertion, giving a total cost of four operations.*

In our basic implementations of SIZE and WORDS, we consider all resource content including HTTP headers and HTML markup. We have also experimented with removal of headers, removal of markup, and removal of both. Our rationale for removing headers is based on our observation in preliminary experiments that these frequently contribute to recrawling of documents that have not changed in their content. For example, header tags such as `Date: Wed, 23 Apr 2003 22:08:08 GMT` reflect the time of retrieval and not the document content, but are updated at each retrieval. Our rationale for removing markup is based on observations that changes in it rarely affect document content and that markup is rarely indexed by search engines.

4.3 Using Document Change in Web Crawling

One of our additional aims in this chapter is to identify change metrics that can use past document change as an effective predictor of future document change. For these purposes we assume that each web resource has been retrieved twice; that is, we are able to determine how much a document has changed and use this as a predictor of significant future change. In practice, document change frequency needs to be predicted after a document has been retrieved only once, and this is something we examine further in Chapters 5 and 6.

Since our aim is to identify change metrics that can predict future change our definition for change metric is altered accordingly.

1. NONE— retrieve all documents at time $t+n$ but none of the documents at time $t+2n$. In this way, all other metrics are guaranteed to perform as well as the NONE metric, since they will contain all documents from time $t+n$, while some subset will be replaced with documents from time $t+2n$. Note that the NONE metric is identical for both predicting future change and detecting change.

2. ALL— retrieve all documents at time $t+n$, and the subset of documents at time $t+2n$ that have changed between time intervals t and $t+n$, that is, any document version d_{t+n} that is not bytewise identical to d_t is recrawled. In this way, any document not updated to time $t+2n$ will be updated to time $t+n$.

3. SIZE— retrieve all documents at time $t+n$, and the subset of documents at time $t+2n$ that have changed by more than α bytes in size between time intervals t and $t+n$.

4. WORDS— retrieve all documents at time $t+n$, and the subset of documents at time $t+2n$ that have changed by more than α words between time intervals t and $t+n$.

With each metric s, we compute a function $C_s(d_t, d_{t+n}, \alpha)$ that evaluates to 0 or 1 for a change threshold α and two versions of document d that have been retrieved at time intervals t and $t+n$. A value $C_s(d_t, d_{t+n}, \alpha) = 1$ indicates that document d has changed by more than α under metric s and should be recrawled, while $C_s(d_t, d_{t+n}, \alpha) = 0$ indicates that no change has occurred or the change is less than the threshold α. We make two simplifying assumptions: first, when $C_s = 1$ we assume that sufficient resources are available to recrawl document d in the next time interval $t+2n$; and second, we assume only binary values for C_s, that is, a priority for recrawling is not available.

An implementation of the function $C_s(d_t, d_{t+n}, \alpha)$ is effective if it is able to identify document changes that will impact the accuracy of the search engine. We discuss how the impact on search engine accuracy is measured in Section 4.4. Moreover, the function $C_s(d_t, d_{t+n}, \alpha)$ is efficient if it minimises the number of resources that are retrieved to those that have an impact on effectiveness. We quantify efficiency in Section 4.5 by measuring the number of documents retrieved in a given time interval, and consider it relative to the extremes of recrawling all documents with any change and not recrawling.

	Document changed	Document not changed
Crawler revisits	SUCCESS	FAILURE
Crawler does not revisit	FAILURE	SUCCESS

Table 4.1: Matrix for determining the success or failure of a crawl in acting on a single document.

4.4 A Measurement Framework for Web Crawling

An obvious measure of the accuracy of a recrawling strategy is to record the success and failure rate for revisiting documents. For example, after using a metric to decide whether to recrawl each document in the collection, the prediction success may be recorded in a confusion matrix that tabulates *successes* and *failures*, as shown in Table 4.1. A success is where a document that was predicted to change was retrieved and had changed, or where a document was predicted not to change and had not changed. Failures are where a document was predicted to not change but did change, and where a document predicted to change and was retrieved but had not changed. The sum of the values in the matrix is the number of documents in the collection. When evaluating change metrics we do not consider new documents, as we are examining change in existing documents. However, later in Chapter 5 we consider new documents that are retrieved during the recrawl process as well as their popularity, and in this case we use a modified evaluation metric, which we highlight in Table 5.1.

For a search engine, we argue that counting successes and failures does not accurately reflect the performance of a recrawling strategy because it does not take into account how often documents are returned to search engine users. As for the use of terms in queries [Spink et al., 2001] and word occurrences in text [Williams and Zobel, 2005], we have observed that the frequency of appearance of documents as highly-ranked answers is skewed. That is, a subset of the documents in the collection is often returned as answers to a stream of queries, while the majority of documents are rarely returned. Figure 4.2 shows this effect for the collection and queries we use in our experiments in Section 4.5. For our set of 50 queries, over 10,000 of our 12,348 documents are never ranked in the top 100 responses. In contrast,

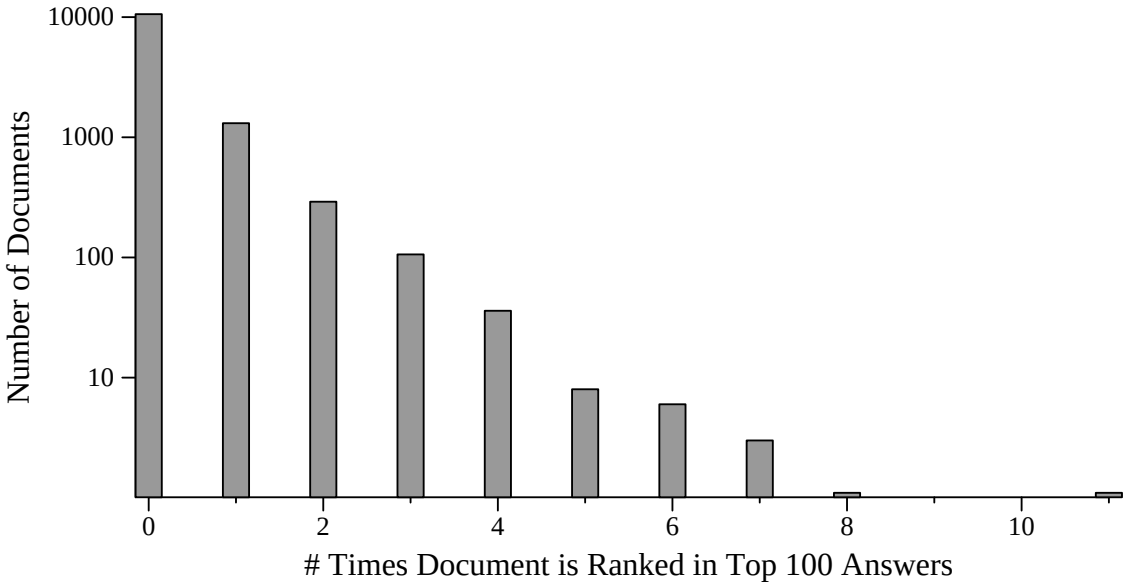


Figure 4.2: Skewed ranking of collection documents in response to 50 queries. The graph shows on the x-axis the number of times a document appears in the top 100 responses to the 50 queries, while the y-axis shows the number of documents with that frequency. For example, over 1,300 documents appear once in the top 100 answers for the 50 queries. Over 10,000 documents are never returned in the top 100 responses (0 on x-axis).

twenty of the documents appear five times or more, and one document appears in answers to eleven queries. Similar access frequencies have also been observed in larger collections [Garcia et al., 2004].

Skewed ranking of documents suggests two principles that need to be considered when recrawling documents. First, recrawling documents that are returned as answers to queries is important, while recrawling unaccessed documents is likely to be unnecessary. Second, a complete recrawling strategy should include document access frequencies, an idea that we explore further in Chapter 5.

Returning to our first principle, we propose a novel crawler evaluation measurement that considers the impact that recrawling has on search performance. Specifically, we determine the answers that would have been returned as the top 1,000 ranked responses to a query if the collection were entirely up-to-date, and then measure how these results are affected when the collection is formed from a recrawling strategy. This process of forming a collection is

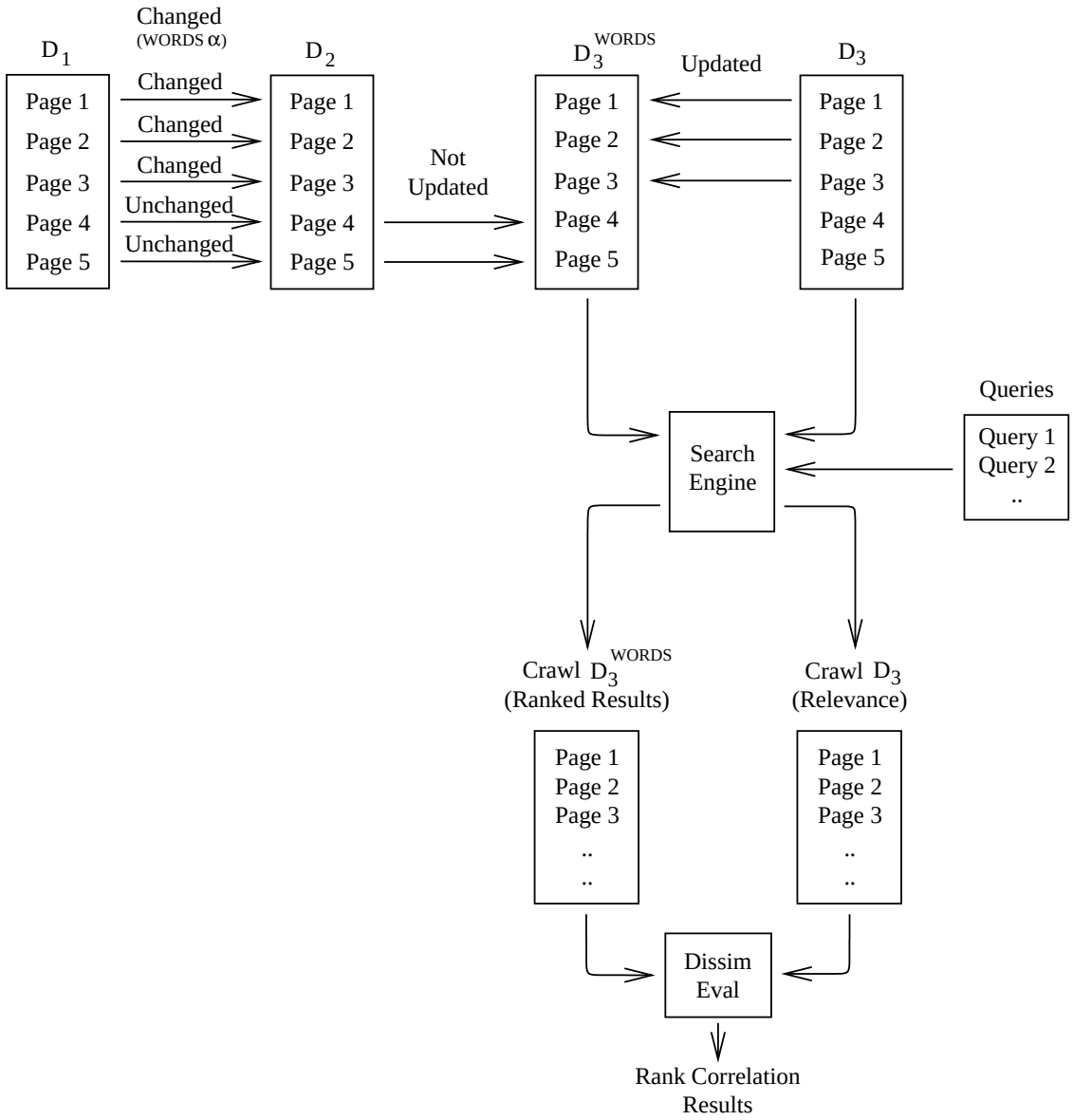


Figure 4.3: Evaluating a change metric's capacity for predicting future change. The WORDS *metric is evaluated using the D_1, D_2 and D_3 collections. Documents that change in the $D_1 \rightarrow D_2$ interval by a significant amount, as determined by* WORDS α*, are updated in the $D_2 \rightarrow D_3$ interval. The partially updated collection D_3^{WORDS} is then compared to the fully up-to-date collection D_3 by using a search engine and examining the ranked search results of queries against the two collection. The Dissim rank correlation metric is used to measure the degree to which the ranked up-to-date D_3 results match the ranked D_3^{WORDS} results.*

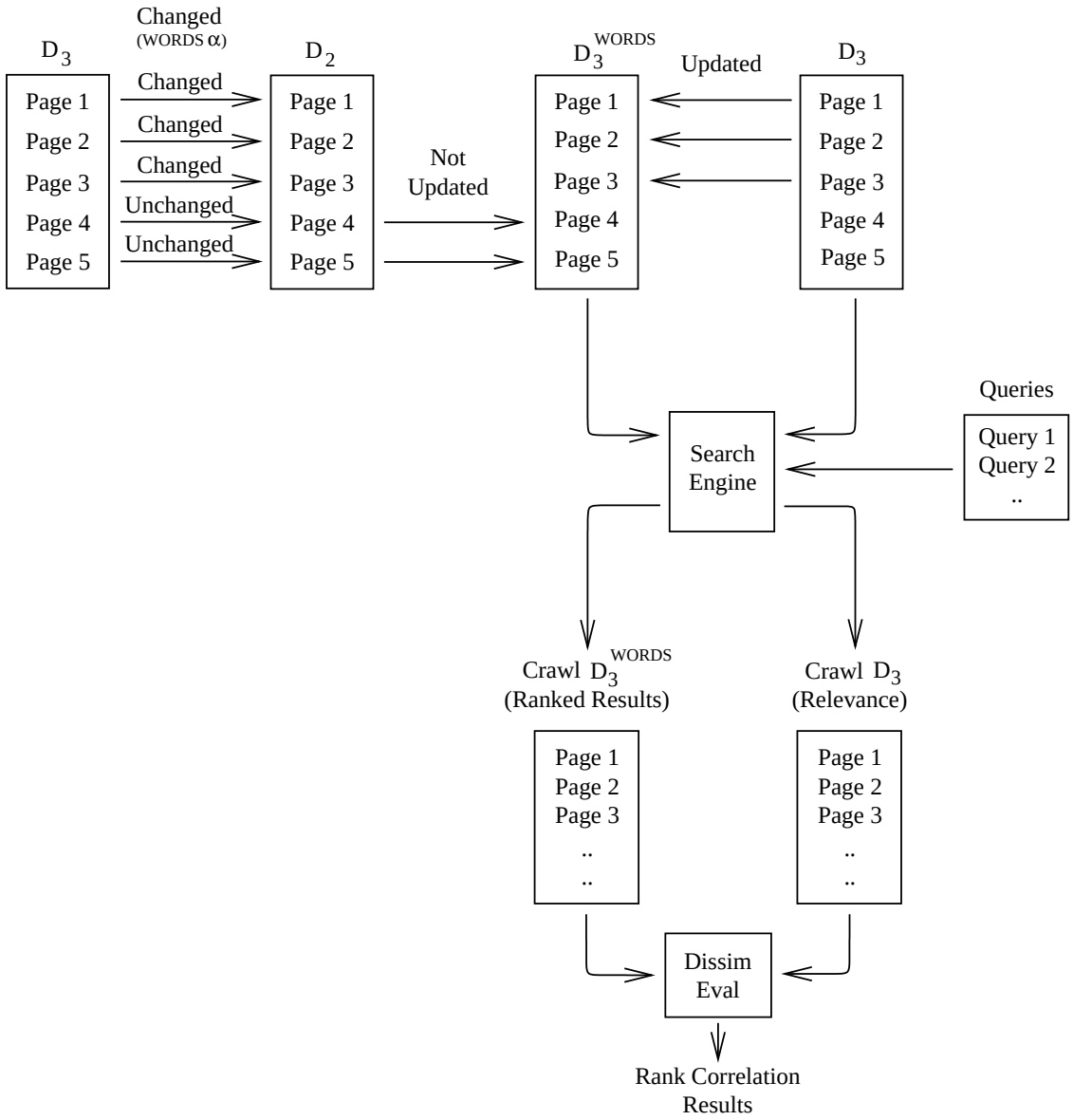


Figure 4.4: Evaluating a change metric's capacity for detecting meaningful change. The WORDS *metric is evaluated using the D_2 and D_3 collections. Documents that change in the $D_2 \rightarrow D_3$ interval by a significant amount, as determined by* WORDS α*, are updated in the $D_2 \rightarrow D_3$ interval. The partially updated collection D_3^{WORDS} is then compared to the fully up-to-date collection D_3 by using a search engine and examining the ranked search results of queries against the two collection. The Dissim rank correlation metric is used to measure the degree to which the ranked up-to-date D_3 results match the ranked D_3^{WORDS} results.*

similar in spirit to the TREC experimental testbed [Harman, 1995]: for a set of queries, a set of known answers are stored, and techniques can be evaluated by measuring how accurate they are in returning the set of known answers. We explain this process below.

We determine the correct answers that would be returned if a collection was up-to-date using four steps. First, we crawled and recrawled a document collection at time intervals t, $t+n$, and $t+2n$ to form three collections D_1, D_2, and D_3 respectively. Second, we extracted fifty queries from a well-studied Excite search engine log [Spink et al., 2001]. Third, we ran the queries using our search engine on the D_3 collection retrieved at time $t+2n$. Last, for each query we stored as the relevant answers either the top 1,000 ranked responses or the number of ranked responses returned, whichever is less. In this way, the performance of our search engine with a perfect recrawling strategy for the time interval is known. (Further details on the collection, queries, and search engine are provided in Section 4.5.)

We can now evaluate the performance of our change metrics outlined in Section 4.3 (page 106). Using the document collections D_1 and D_2 crawled at time intervals t and $t+n$, we can compute the change measure $C_s(d_t, d_{t+n}, \alpha)$ for all documents in D_2. For documents where $C_s = 1$, we update the collection D_2 so that the document version d_{t+2n} replaces d_{t+n} to form document collection D_3^{METRIC}, that is, D_3^{WORDS}, D_3^{SIZE}, $D_3^{SHINGLING}$, D_3^{ALL}, and D_3^{NONE} for each metric. Having derived our new collection based on a simulated recrawl strategy, we run our queries on the D_3^{METRIC} collection and record the top 1,000 responses (or the number returned if this is less).

To express the effectiveness of each recrawl metric, we use rank correlation to determine the similarity of the ranked results returned from search D_3^{METRIC} and the correct results from searching D_3. In our approach, a correct answer is one that has the same URL. That is, we view a match as correct even if the document has changed, as long as the correct URL is identified. This approach is logical: though the content of a document may have changed, its content is still ranked as relevant as far as users' queries are concerned, and so, the changes are not important. We measure the similarity of the top 1,000 results (or as many as are returned) using the Dissim rank correlation metric which we outlined in Section 2.4.2 (page 47). In the situation where there are 1,000 relevant answers in the collection and all 1,000 answers are returned in the same order, the rank correlation is 100%. We average the

rank correlation results across all queries and normalise the score using the ALL and NONE results as the maximum and minimum values respectively.

In Figure 4.3, we illustrate the evaluation process further using the document collections D_1 and D_2 crawled at time intervals t and $t+n$. We compute the change measure $C_s(d_1, d_2, \alpha)$ for all documents in D_2 using the WORDS metric and use this to determine which documents to update in the *next* time interval $(t+n,\, t+2n)$, thus allowing us to evaluate how well the WORDS metric can *predict* change.

In contrast, Figure 4.4, illustrates the evaluation process using the document collections D_2 and D_3 crawled at time intervals $t+n$ and $t+2n$. We compute the change measure $C_s(d_2, d_3, \alpha)$ for all documents in D_3 using the WORDS metric and use this to determine which documents to update in the *same* interval $(t+n,\, t+2n)$, thus allowing us to evaluate how well the WORDS metric can *detect* change.

4.5 Results

In this section, we present the results of using the metrics discussed in Section 4.3 to predict when resources should be recrawled. Our experiments were carried out using the public domain search engine Zettair [2008] that implements a variant of the Okapi BM25 ranking function [Robertson and Walker, 1999]. When evaluating the queries, no stopping, stemming, or other techniques were used. Furthermore, the entire documents, including the headers, were used and so all documents contained changes of some degree. We begin by describing the collection and queries we used in this set of experiments.

4.5.1 Collection and Queries

Our PROXY document collection is derived from web resources retrieved by staff and students of the CS department at RMIT University. We obtained a proxy cache log of around 50 MB that lists around 300,000 web resources retrieved by several hundred user agents over a two day period, and used this to guide the retrieval of documents to form a collection. Because of its size, the number of user agents, and observed diversity, we believe that this proxy log represents a realistic source of documents that would form part of a search engine collection. To test this we selected a different subset of queries from the same proxy log and determined that 77% of reachable documents were indexed by Google [2008].

To develop a collection from the list of web resources in the proxy cache log, we carried out the following process:

1. We extracted all URLs that were successfully retrieved (those that had a "`2xx`" HTTP response code) and had a `Content-Type` (a MIME type) of `text/html`; our search engine indexes only HTML documents, and recrawl schemes for other document types — such as PDF, PostScript, RTF, and proprietary formats — are outside the scope of our work. As we outlined in Chapter 2, we only consider HTML documents since they contain an extensive link structure, make up a large proportion of crawled documents, are well studied, and are very different to other document formats in terms of dynamics [Douglis et al., 1997; Wills and Mikhailov, 1999b; Zander et al., 2003]

2. We removed any query strings from URLs so that the base URL was retrieved. Our rationale is that the URLs of dynamically generated documents can be extracted from web documents during a crawl process, but that these are not necessarily the same URLs as would be found in a proxy log; we therefore chose to omit dynamically generated documents from these experiments. For example, the URL:

   `http://www.google.com/search?hl=en&q=Lara+Croft`

   becomes:

   `http://www.google.com/search`

3. We then removed duplicate URLs, and selected every fifth URL to give a list of around 14,000 URLs.

4. We then retrieved the HTTP headers for each URL to ensure the resource was able to be found (that is, to ensure a "`2xx`" HTTP response code was still possible), and deleted any URLs that were no longer accessible.

5. For the remaining URLs, we retrieved the HTML document and HTTP headers and stored these as a "day one" collection.

6. Each day for fourteen further days, we repeated the crawling process and stored the HTML and HTTP headers; using this approach, the minimum time interval we were able to experiment with was one day.

7. For any URL that was not retrieved on all fifteen days, we removed the resource from our fifteen collections. We also removed any document that contained explicit or offensive terms using the script used to filter the TREC VLC collection [Hawking et al., 1998].

The resulting collection consisted of fifteen copies — one each from fifteen consecutive days — of 12,348 documents. Its total size on day one was 126 MB. Of these 12,348 documents, only 4,780 — or 38.7% — contained either a `Last-Modified` or an `Expires` HTTP header.

In our experiments, we used fifty queries extracted from a query log from the Excite search engine [Spink et al., 2001]. To ensure that the queries had answers in our collection, we selected the first 50 queries that returned at least 500 answers with any of the query terms and had at least 10 answers that contained all of the query terms; these heuristics were chosen based on empirical observation with the aim of finding queries that had relevant answers in our collection. Furthermore, while we ensured that the queries we used had at least 500 answers, we computed rank correlation over 1000 documents, to avoid possibly skewing results by ignoring documents that were ranked highly to more constrained query topics, hence improving the amount of variety among documents that were considered as part of the study. We also experimented with using only the top 30 documents to determine the impact on highly ranked results.

We use the Proxy collection exclusively in this chapter for the purposes of evaluating change metrics. It represents a general collection of pages retrieved from many different domains however it contains no link information. This collection allows us to evaluate the effectiveness of our change metrics on a general Web collection. Throughout all our experiments in later chapters we use a second collection, ABC derived from a single domain, which we described in greater detail in Section 3.2. This collection contains additional information such as link structure and anchor text which make it suitable for our experiments in later Chapters. In order to ensure the suitability of our change metrics we also evaluate them against this collection.

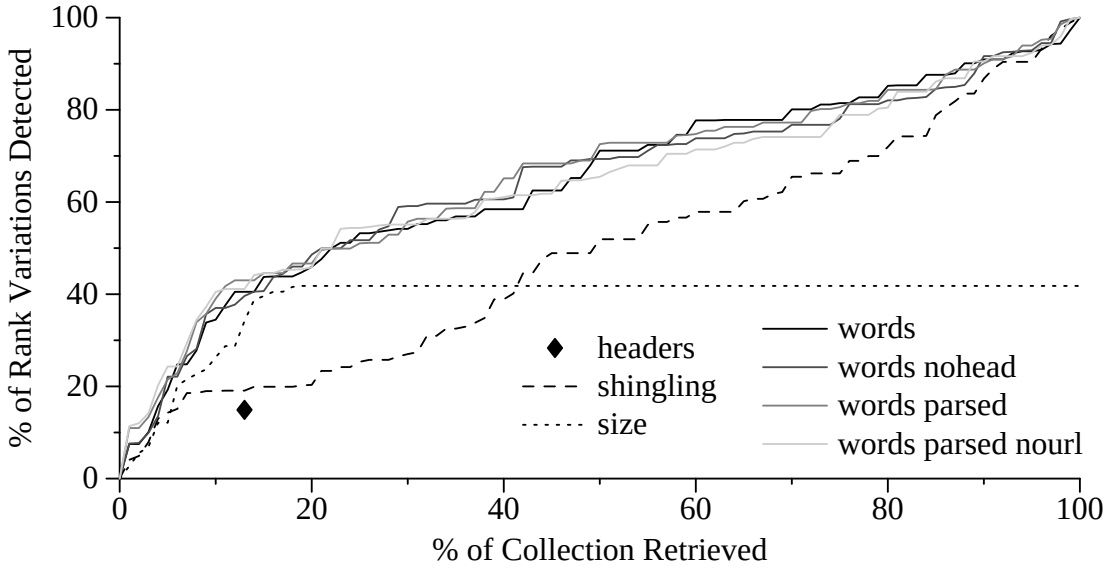


Figure 4.5: Change prediction over an interval of one crawl on the PROXY *collection. Crawl one and two are used to predict changes on crawl three. The x-axis indicates the percentage of the collection that is updated while the y-axis indicates the percentage of variations in rank correlation that are detected, as the α change threshold is altered.*

4.5.2 Efficiency and Effectiveness

So far in this section, we have highlighted how we formed our collection and queries. Next, we present our experimental results for our various metrics against this collection and queries. First we present the effectiveness of our schemes for *predicting future* change using the PROXY and ABC collections. Later we examine the effectiveness of our scheme for *detecting current* change against these same collections.

Predicting Future Change

In Figure 4.5 we plot the overall effectiveness of each metric for predicting change using the PROXY collection. The y-axis indicates the average rank correlation between the query results for collection D_3^{METRIC} and D_3, normalised against the ALL and NONE metrics. That is, 0% and 100% on the y-axis represent the NONE and ALL metrics respectively. The x-axis indicates the percentage of the collection that is updated as the threshold for change α is reduced for each metric.

The results shown are for predicting which documents to retrieve on the third of three consecutive crawls, that is, documents are retrieved on crawl one and two, and predictions using different change metrics are used to decide which documents to retrieve on crawl three. We performed the same experiment with crawl twelve, thirteen, and fourteen and found the same relative results. All document content — including headers and markup — is used in the change detection process for the WORDS, SHINGLING and SIZE metrics. The HEADERS metric uses only the HTTP header information to predict which documents will change. The WORDS NOHEAD metric removes the HTTP headers prior to computing the WORDS change value. The WORDS PARSED metric removes HTML markup prior to computing the WORDS change value. Finally, the WORDS PARSED NOURL metric removes both HTML markup and URL link information prior to computing the WORDS change value.

The results in Figure 4.5, show that our metrics (SIZE and WORDS) are efficient and effective. After retrieving 15% of the entire collection, the WORDS metric detects over 43% of the variations in rank due to stale documents, while the SIZE metric detects over 39%. In contrast, the SHINGLING approach detects less than 20% of these same variations. Interestingly, the SHINGLING approach must retrieve over 41% of the collection – about 2.7 times the amount of documents — to produce the same degree of collection consistency as the WORDS metric. The WORDS metric achieves this level of performance with an α change threshold of 24 words.

We note that the additional preprocessing techniques, such as the removal of HTTP headers, parsing of HTML and removal of URLs appears to have little affect on the overall performance of the WORDS scheme. This suggests that the WORDS scheme is robust enough to handle their inclusion and, as we stated earlier, changes in markup rarely affect document content and markup is rarely indexed by search engines.

Furthermore, it is clear that the SIZE metric cannot detect *all* changes, since changes in content may not affect the overall size of the document. In this case, the SIZE metric detects a maximum of about 42% of all rank variations.

Finally, the performance of the HEADERS approach shows that the use of the Last-Modified and Expires HTTP headers for predicting change is largely ineffective. It detects less than 15% of rank variations, while retrieving 13% of the collection.

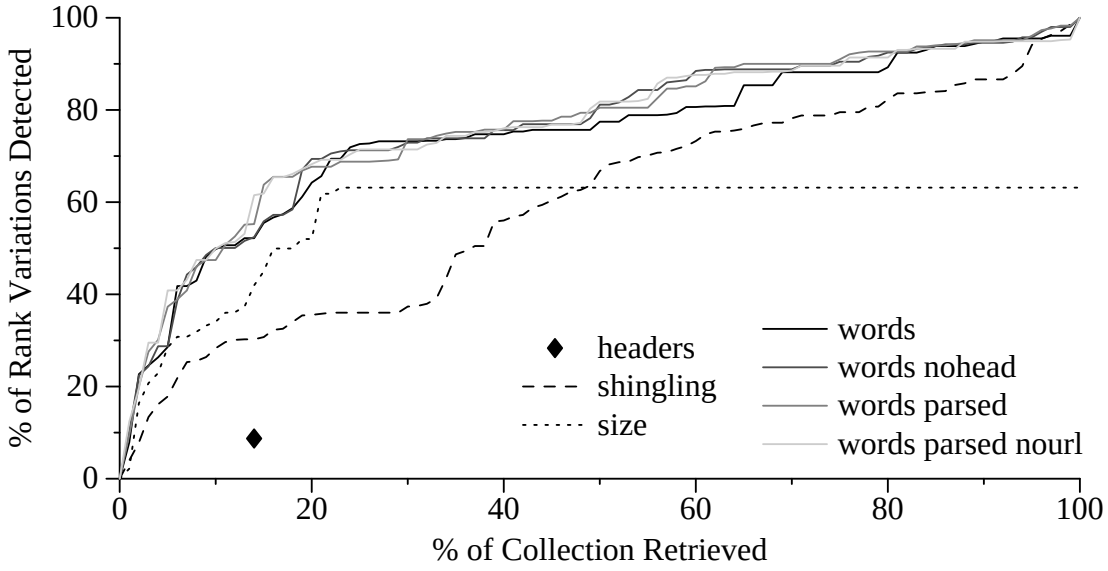


Figure 4.6: Change prediction over an interval of three crawls on the PROXY *collection. Crawl one and four are used to predict changes on crawl seven. The x-axis indicates the percentage of the collection that is updated while the y-axis indicates the percentage of variations in rank correlation that are detected, as the α change threshold is altered.*

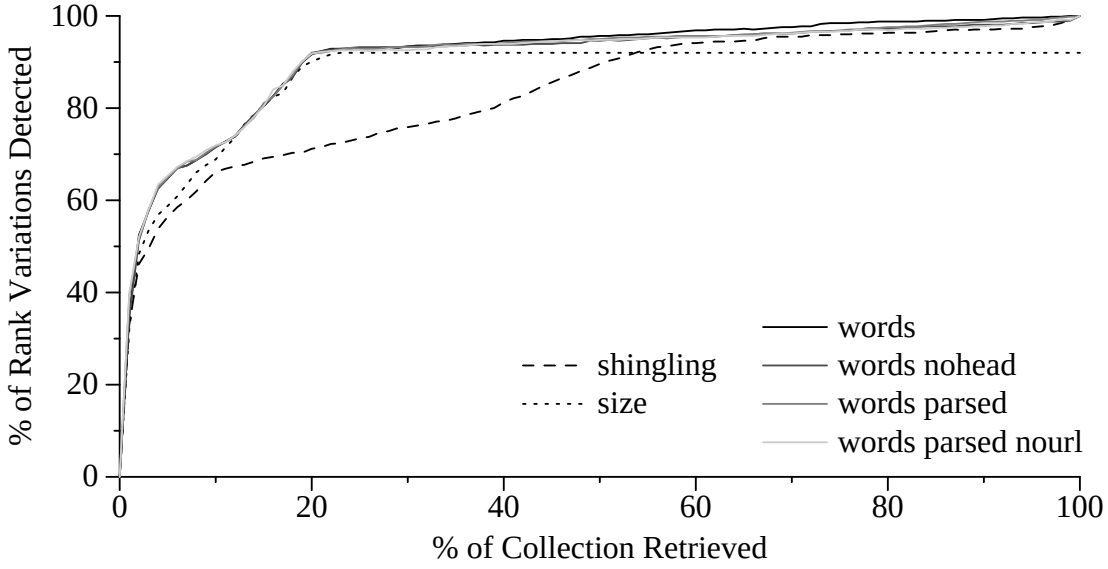


Figure 4.7: Change prediction over an interval of one crawl on the ABC *collection. Crawl one and two are used to predict changes on crawl three. The x-axis indicates the percentage of the collection that is updated while the y-axis indicates the percentage of variations in rank correlation that are detected, as the α change threshold is altered.*

In Figure 4.6 we plot the effectiveness of each scheme at predicting change using the PROXY collection, this time over three crawl intervals, that is, documents are retrieved on crawl one and crawl four. Changes detected by different metrics are used to determine which documents to update on crawl seven. In this graph we see that the pattern in the relative performance of each metric is similar to that in Figure 4.5, with the WORDS metric and its variations performing best. The WORDS metric detects nearly 72% of all changes in rank while retrieving just 24% of the collection with an α change threshold of 13 words. The SHINGLING approach detects just over 36% of the variation in the rank of search results for the same number of documents retrieved, while the SIZE metric detects its maximum at just over 63%. Finally, the HEADERS approach detects just 9% of rank variations while retrieving 14% of the collection.

In the next set of graphs we examine the change prediction performance of our schemes on the ABC collection. In Figure 4.7, we plot the effectiveness over intervals of one crawl on the ABC collection. We note that the WORDS and SIZE metrics easily outperform the other metrics at predicting change over one crawl intervals. The WORDS and SIZE metrics detect nearly 93% and 92% of variations, respectively, while retrieving 22% of the collection. In contrast the SHINGLING approach detects less than 73% of variations after retrieving the same number of documents. To detect the same amount of variations as the WORDS metric, the SHINGLING metric must retrieve more than 55% of the collection — 2.5 times the number of documents retrieved by the WORDS metric. The WORDS metric achieves this with an α change threshold of 12 words. The different variations of the WORDS metric have virtually no impact on its performance.

Once again we note that the SIZE metric is restricted in its ability to detect variations, this time to a maximum of 92% of all rank variations. Finally, we note that we have not included any HEADERS results for the ABC collection. After examining the headers, we observed that the HTTP headers had been modified by administrators of the ABC domain to automatically expire just under a week after they were retrieved — one of the problems highlighted previously in Sections 2.1.1 (page 25) and 2.8.3 (page 73).

Another interesting observation that can be made from our results is that not all documents and changes are created equal. As we highlighted earlier in Section 4.4, query results are skewed. Hence, updating certain documents can have a greater impact on search results.

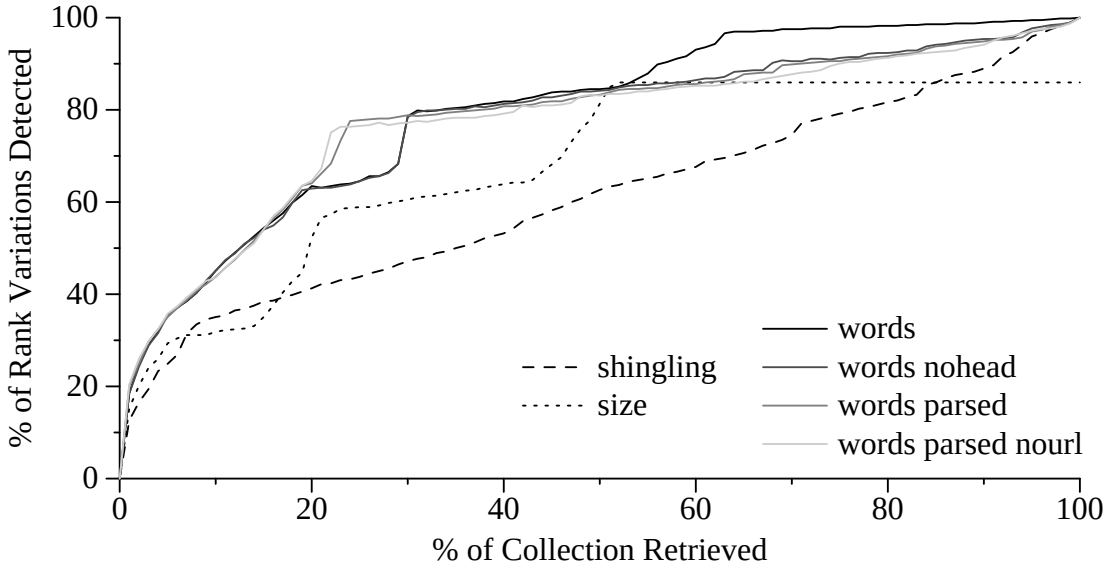


Figure 4.8: Change prediction over an interval of three crawls on the ABC *collection. Crawl one and four are used to predict changes on crawl seven. The x-axis indicates the percentage of the collection that is updated while the y-axis indicates the percentage of variations in rank correlation that are detected, as the α change threshold is altered.*

This explains why improvements in rank correlation are not linear. That is, in our ABC collection 87,351 documents change by any amount in content over the crawl two and crawl three interval. This composes 22.7% of the entire collection, however it is possible to capture the majority of these variations (64%) by updating just 5% of the entire collection — just 19,249 documents out of a total of 384,992. If each document had an equal impact on rank, achieving a rank correlation of 64% would require 64% of all changed documents to be updated — 55,904 out of a total of 87,351 changed documents instead of just 19,249.

Finally, in Figure 4.8, we present the results of predicting change over intervals of three crawls using the ABC collection. We note that once again the Words metric and its variations produce the best results. Interestingly, the parsed variations Words Parsed and Words Parsed NoURL perform better than the Words metric during the interval from 18% to 31% of the collection retrieved, however from 31% onwards the Words metric outperforms all other Words variations. The cause of this variation is due to modifications to markup in a number of documents on crawl four of the Proxy collection.

At 31% of the collection retrieved, the WORDS metric detects almost 80% of all variations in rank, with an α change threshold of 13 words. At the same percentage of the collection retrieved, the SIZE and SHINGLING metrics predict about 61% and 48% respectively. The SIZE metric is limited to a maximum of 86%.

Throughout our experiments into *predicting future* change we show that the WORDS metric and its variants consistently outperform the SIZE and SHINGLING approaches. The basic WORDS metric appears to be effective at predicting changes that will affect rank results. Furthermore, an α change threshold between 12 and 24 words appears to capture the majority of these changes. Next we present the effectiveness of our schemes for *detecting* change using the PROXY and ABC collections.

Detecting Current Change

In this section we examine the effectiveness of our change metrics at *detecting* changes that affect ranked results in the PROXY and ABC collections.

Our first set of results, in Figure 4.9, show the effectiveness of our schemes at detecting changes over a one crawl interval, beginning on crawl one — changes between crawl one and two are used to update documents on crawl two. The α change threshold is adjusted to update a subset of the collection on crawl two, which is then compared to the up-to-date crawl two collection, as highlighted in figure 4.4. In this first graph we note that the WORDS metric and its variants are most effective. The WORDS metric detects close to 97% of rank variations while updating just 22% of the collection, using an α change threshold of 10 words. For the same number of updates, the SIZE metric achieves its maximum effectiveness of 84%, whereas, the SHINGLING approach detects less than 63% of the variations in rank. The SHINGLING metric must update more than 76% of the collection — almost 3.5 times as many documents — in order to achieve the same level of effectiveness as the WORDS metric. Interestingly, the HEADERS approach is much more effective at detecting change than it is at predicting change. Here we see that HEADERS detects almost 56% of variations in rank while retrieving 9% of the collection. On closer inspection, the majority of these changes are detected by the expires HTTP header. We note that the entire collection must be updated before all rank variations are detected due to changes in the HTTP headers. Furthermore, we

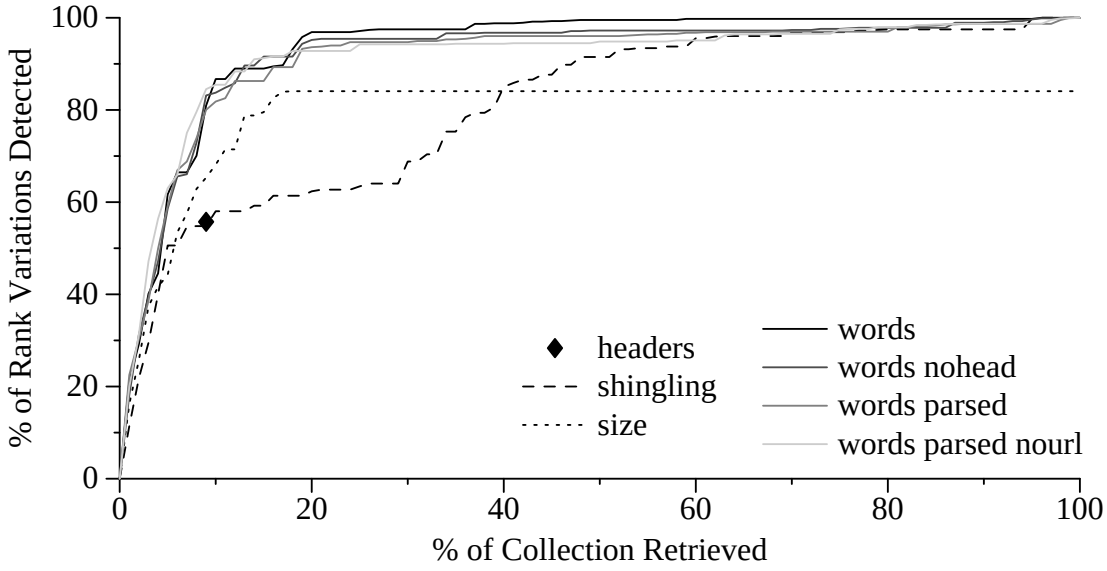


Figure 4.9: Change detection over an interval of one crawl on the PROXY *collection. Crawl one and two are used to decide pages to update on crawl two. The x-axis indicates the percentage of the collection that is updated while the y-axis indicates the percentage of variations in rank correlation that are detected, as the α change threshold is altered.*

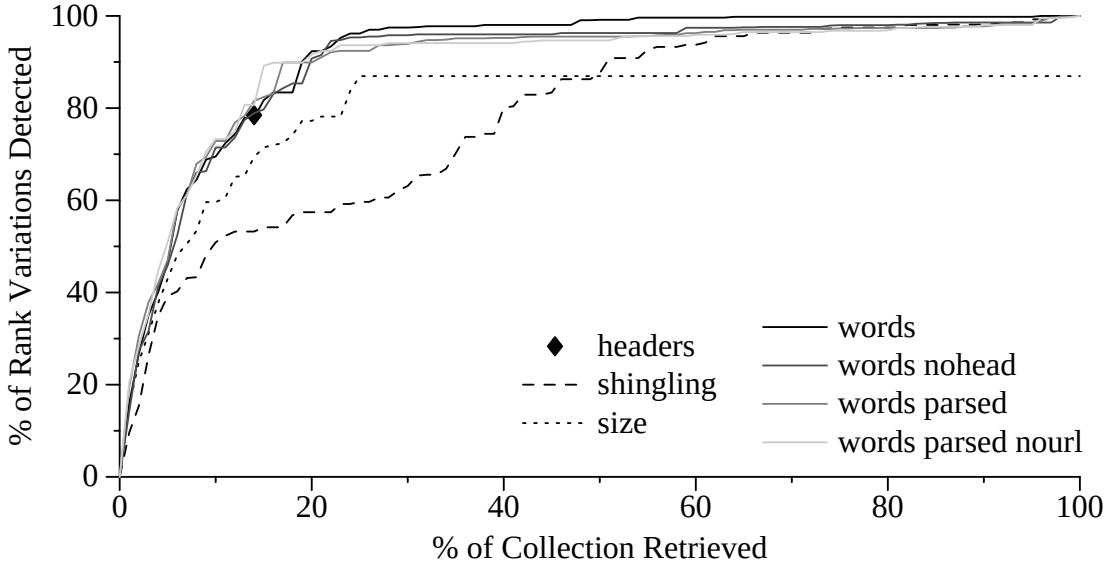


Figure 4.10: Change detection over an interval of seven crawls on the PROXY *collection. Crawl one and seven are used to decide pages to update on crawl seven. The x-axis indicates the percentage of the collection that is updated while the y-axis indicates the percentage of variations in rank correlation that are detected, as the α change threshold is altered.*

index the entire document — including the HTTP headers when evaluating rank variations. Once documents are parsed only 38% of the collection contains changes.

In Figure 4.10, we examine the PROXY collection over an interval of seven crawls, using changes in documents from crawls one and seven. We note that once again, the WORDS metric and its variants detect the largest percentage of rank variations. For a brief period, from about 13% to 19% of the collection updated, there is a spike in the performance of the parsed HTML variations of the WORDS metric, however, on the whole, the basic WORDS metric performs best. With an α change threshold of 9 words, the WORDS metric retrieves about 28% of the collection while detecting about 98% of rank variations. In contrast, the SIZE and SHINGLING approaches detect about 87% and 61% respectively. The SHINGLING metric must update 75% of the collection to achieve the level of consistency in rank results that the WORDS metric achieves with just 28% of the collection. Finally, the HEADERS approach produces its best result during this seven crawl interval, detecting 78% of variations in rank while retrieving 14% of the collection. Examining the results more closely reveals that the last modified HTTP header has a greater impact in this case. The large improvement over the one crawl interval suggests that the changes detected over shorter intervals are not significant as those over longer intervals. With the removal of markup using the WORDS PARSED scheme, we note that 41% of the collection contains changes.

In Figure 4.11, we present the effectiveness of our schemes at detecting changes over a one crawl period on the ABC collection. The performance of all metrics, except the SHINGLING approach, is on par with that of the basic WORDS metric. Once again the SIZE metric is restricted in its ability to detect all changes — this time to about 94% of rank variations, which it achieves after updating 24% of the collection. The WORDS metric detects just over 93% after updating about 20% of the collection, while the SHINGLING metric detect just 70%. To achieve the same effectiveness as the WORDS metric at 20%, the SHINGLING metric must update 54% of the collection — 2.7 times the number of documents as that of the WORDS metric. The parsing of the collection reduces the number of changed documents to 53%.

In Figure 4.12, we highlight the effectiveness of our schemes over a seven crawl interval on the ABC collection. Here we see that the WORDS PARSED and WORDS PARSED NOURL metric perform slightly better than the WORDS and WORDS NOHEAD metrics until 28% of the collection is updated. From this point onwards, the WORDS metric outperforms WORDS

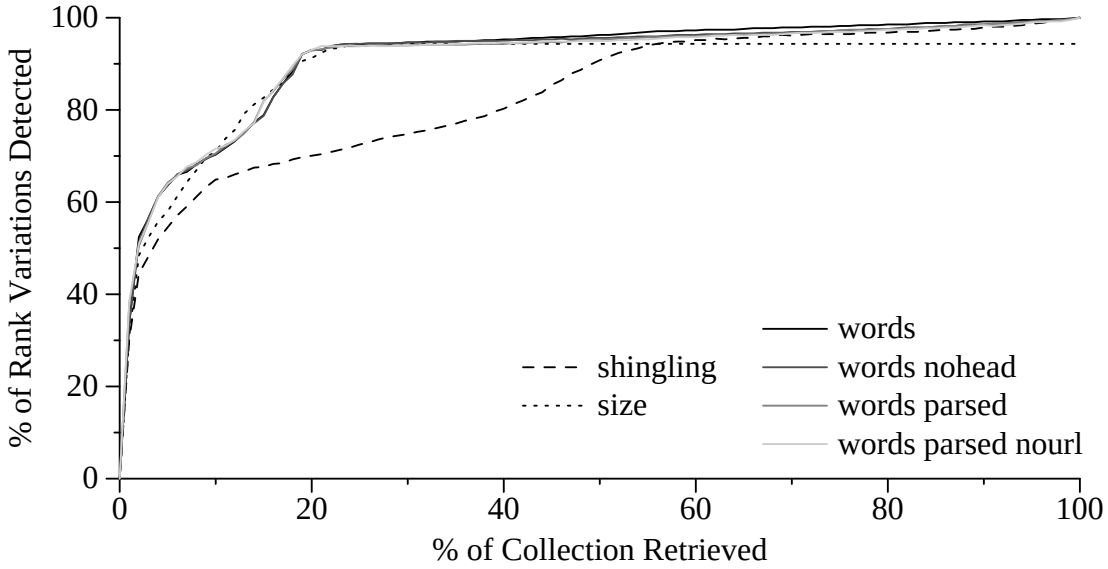


Figure 4.11: Change detection over an interval of one crawl on the ABC *collection. Crawl one and two are used to decide pages to update on crawl two. The x-axis indicates the percentage of the collection that is updated while the y-axis indicates the percentage of variations in rank correlation that are detected, as the α change threshold is altered.*

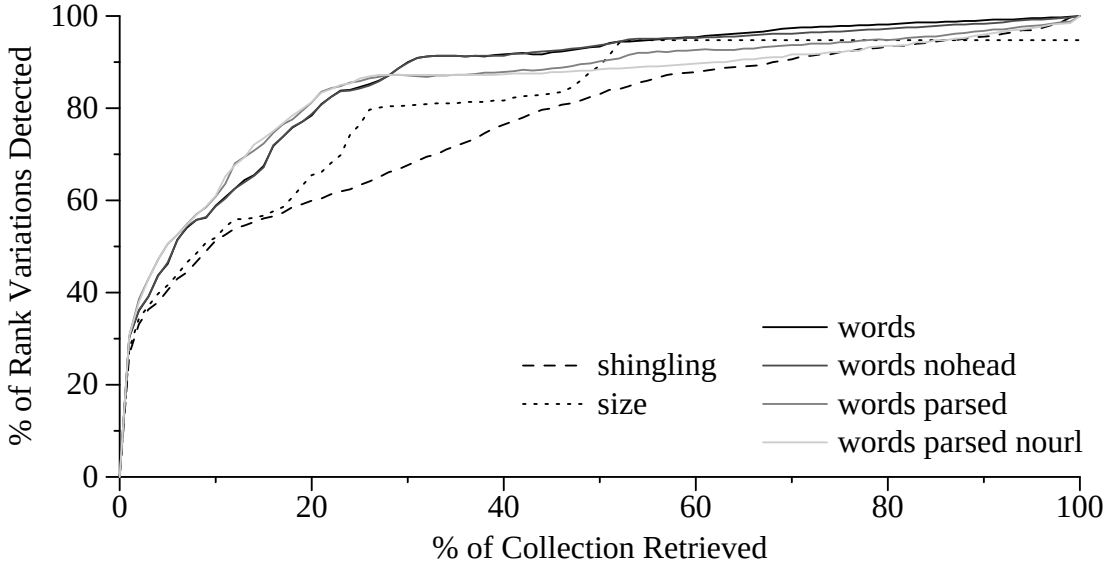


Figure 4.12: Change detection over an interval of seven crawls on the ABC *collection. Crawl one and seven are used to decide pages to update on crawl seven. The x-axis indicates the percentage of the collection that is updated while the y-axis indicates the percentage of variations in rank correlation that are detected, as the α change threshold is altered.*

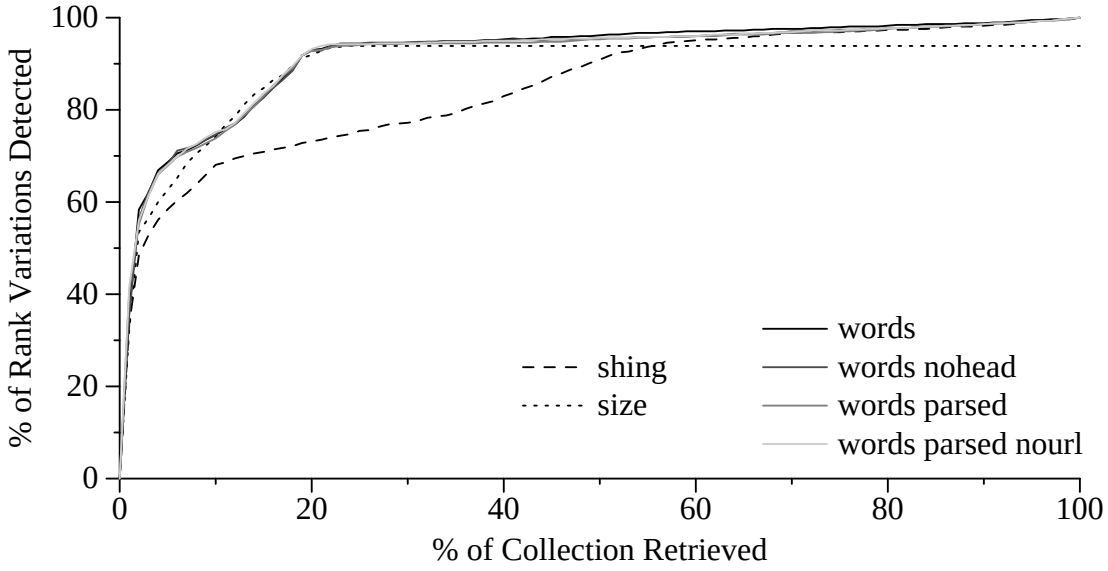


Figure 4.13: Change detection over an interval of one crawl on the ABC *collection using top 30. Crawl one and two are used to decide pages to update on crawl two. The x-axis indicates the percentage of the collection that is updated while the y-axis indicates the percentage of variations in rank correlation that are detected, as the α change threshold is altered.*

Parsed and Words Parsed NoURL metrics. With an α change threshold of 12 words, the Words metric updates 33% of the collection, while detecting over 91% of rank variations. In contrast, the Size and Shingling metrics detect roughly 81% and 70% respectively. In order to be as effective as the Words metric after updating 33% of the collection, the Size and Shingling metrics must update 51% and 72% respectively. Furthermore, the Size metric cannot detect more than 95% of all rank variations. Finally, we note that 54% of the collection contains changes after parsing the documents.

Our results so far have examined the effectiveness of our schemes using the top 1000 query results — a design choice based on our observation that search engines mainly retrieve 1000 results and TREC evaluation set-ups only consider the top 1000 results. However, as we highlighted in Section 2.4.1 (page 44) most search engines return only ten results per page of results by default [Google, 2008; Yahoo!, 2008; Ask.com, 2008; Altavista, 2008; Lycos, 2008] and many users do no look past the first page of results [Jansen et al., 1998; Joachims, 2002; Lempel and Moran, 2003; iProspect, 2006].

Crawl	1-2 vs 2-3		1-4 vs 4-7	
Collection	ABC	PROXY	ABC	PROXY
WORDS α Threshold	12	24	13	13
Updated	22%	15%	31%	24%
Effectiveness	93%	44%	80%	72%

Table 4.2: Performance of WORDS *metric at predicting change with particular α change thresholds and crawl intervals.*

Crawl	1-2 vs 1-2		1-7 vs 1-7	
Collection	ABC	PROXY	ABC	PROXY
WORDS α Threshold	21	10	12	9
Updated	20%	22%	33%	28%
Effectiveness	93%	97%	91%	98%

Table 4.3: Performance of WORDS *metric at detecting change with particular α change thresholds and crawl intervals.*

In the final result in this section, presented in Figure 4.13, we demonstrate the effectiveness of our scheme using the top 30 query results instead. Interestingly, the performance is virtually identical to that over 1000 results, presented earlier in Figure 4.11. While similar observations were made for the other result sets, we have not included these for brevity.

Throughout our experiments into *detecting* change we show that the WORDS metric and its variants consistently outperform the SIZE and SHINGLING approaches. As with *predicting future* change, the basic WORDS metric can consistently detect changes that will affect rank results. Furthermore, an α change threshold between 10 and 21 words appears to capture the majority of these changes.

In Tables 4.2 and 4.3, we summarise the α change thresholds which efficiently predict the majority of future changes, and detect the majority of changes, respectively.

Based on our results in this section, we conclude that the WORDS metric is a highly effective and efficient method for distinguishing meaningful change for web crawling purposes, motivating us to use it in our work in later chapters. Throughout our work in later chapters,

we use the WORDS metric with an α change threshold of 10 to ensure that the majority of meaningful changed are captured.

4.6 Summary

In this chapter we have proposed various change metrics and have shown their relative performance. We have also proposed an effective measure for determining the impact of change and recrawling on user search results. In later chapters we use these methods as part of our experiments.

Web crawling is essential to the maintenance of an up-to-date and complete document collection for a web search engine. We have proposed simple measures to determine when document content has changed and will affect the search process. Our approaches are based on comparing document versions, and will form an important component of an overall recrawl evaluation strategy.

Our results show that the use of the number of words that change in a document is a practical tool for guiding document recrawling and an effective way of detecting changes that will affect user search results. For example, by recrawling just 22% of the PROXY collection, the WORDS metric can update a collection such that only 3% of search results are inaccurate over a one crawl period. In contrast, using the SHINGLING approach has an inaccuracy of 37% for the same number of documents updated. Furthermore, the SHINGLING metric must retrieve 3.5 times as many documents to achieve the same level of accuracy in search results as the WORDS scheme.

Despite its inability to detect all changes, our SIZE metric manages to keep search results relatively consistent, failing to accurately maintain just 16% of search results over a one crawl period on the PROXY collection. In contrast, the HTTP headers detect only a fraction of the changed resources, and fail to accurately maintain 44% of ranked results.

We conclude that our WORDS metric is an effective and efficient tool for both guiding web recrawling for web search engines and detecting meaningful changes in documents. Furthermore, the WORDS metric detects changes more effectively than the SHINGLING approach commonly used by other work. Over a one crawl interval, the WORDS approach can achieve the same performance as the SHINGLING approach while retrieving one third of the number

of documents resulting in substantial savings in crawler bandwidth. The WORDS metric is an effective and efficient approach for finding significant changes in documents.

The evaluation metrics discussed in this chapter form only one part of an effective and efficient recrawling strategy and we examine other aspects as well. In the next chapter we investigate ongoing trends for large collections, that is, how a priority queuing strategy can be used to guide efficient and effective querying. We also investigate the use of automatic document categorisation through the use of anchor terms and its use in predicting recrawl frequency in Chapter 6.

Chapter 5

Stateless Crawl Ordering Schemes

As we described in Chapter 1 (page 3), search engines are an important tool for many users who seek to resolve their information needs, in both a general Web environment and within an enterprise environment. While there has been research into recrawling changed documents with the aid of a long history of change statistics, there has been less investigation of maintenance of collection freshness when these statistics are not available. In this chapter we investigate strategies for crawl ordering that aim to recrawl documents that have changed and avoid recrawling static documents, while utilising only the information available after one crawl. Our results, based on the ABC and CSIRO domains, show that, while the appropriate crawling strategy is consistent for a particular domain, it can vary between domains depending on their topology and dynamics. Furthermore, while many of the crawl ordering schemes we test have been shown to have good properties in terms of retrieving important documents early in a crawl, they are not as well suited to recrawling changed documents. In this chapter we also investigate popularity-based metrics. We conclude that a static crawl ordering scheme alone is not sufficient for maintaining index consistency and coherence, motivating our work in Chapter 6.

5.1 Crawler types

While there has been research into *static* crawl ordering schemes to evaluate their ability to retrieve "good" documents early in a crawl, no studies to our knowledge have evaluated their performance with regard to recrawling changed documents. We describe the schemes

that we evaluate in this section as being static, as opposed to adaptive, since they do not incorporate any change statistics into the crawling process.

Furthermore, while many studies have examined crawling schemes that incorporate a long history of change, few studies have examined schemes for the cases when this information is unavailable. We describe these schemes as also being stateless, since they are applied to a web collection after only one crawl has been conducted, and hence there are no past change statistics with which to predict future change. Stateless crawling algorithms are applicable in cases where the Web has only been crawled once, or new documents must be recrawled for the first time. They have also been used in cases where collection "quality" is considered important [Cho et al., 1998; Najork and Wiener, 2001; Baeza-Yates et al., 2005].

In Section 2.9 (page 78), we described work that evaluates the effectiveness of many standard crawl ordering schemes, in regard to their ability to improve collection quality, as measured by PageRank. In particular, the work we described compares the breadth-first, inlink, PageRank, larger sites first, and On-line Page Importance Computation (OPIC) ordering schemes, finding that many of these schemes perform reasonably well. However, none of these schemes have been examined in relation to their effectiveness at recrawling changed documents.

An additional aspect we explore, that has been largely ignored in past work, is the use of query skew and document popularity. That is, as we highlighted in Section 4.4 (page 108), there is a skew in the frequency that documents are returned as highly-ranked query results: a small subset of documents are returned frequently in response to user queries, while a much larger set are rarely returned.

In this chapter we exploit this query skew from a crawling perspective, incorporating document access frequencies as part of both the crawling strategy and evaluation, to determine its utility. We also investigate the effectiveness of typical static crawl ordering schemes with regard to recrawling changed documents.

In the next section we highlight the way in which these stateless crawl ordering schemes are applied to web collections to evaluate their effectiveness. We follow with a description of the various static crawl ordering schemes that we evaluate in this chapter, including the Inlink, Outlink, Hub, URLlength, URLdepth, Alpha, Query, Large, and Random

Algorithm 5.1: Simple crawl scheme.

Input: List L of all documents d from a previous crawl, an empty list V

```
1: Sort L by some ranking scheme
2: while L is not empty and bandwidth is available do
3:     Visit the first document d in list L
4:     Remove d from L and insert into V
5:     S ← documents linked to by d
6:     foreach s ∈ S do
7:         if s is not in L or V then
8:             Add s to the front of L
9:         end
10:    end
11: end
```

schemes. We also briefly reintroduce the Breadth, Depth, and PageRank schemes, which we covered in Chapter 2.2.2 (page 40).

5.2 Methodology

When evaluating crawl ordering and collection quality, the crawl frontier is ordered using the scheme to be tested, such as depth-first order, and then, for each document that is retrieved, the quality of the collection is determined using a metric, such as PageRank. In Section 2.9 (page 78) we highlighted several past studies that have used this methodology, however, when evaluating crawl ordering and collection freshness, the approach that we use is somewhat different.

During the initial crawl the entire site is crawled, since a copy of all documents is required for indexing purposes. These documents, their URLs, and their link structure are then used by the schemes to rank documents for the next crawl of the site. Each of the schemes we evaluate operates with nothing more than the statistics available after only one crawl. Some schemes, such as PageRank and inlink, use this to order the next crawl, while other schemes, such as breadth-first and depth-first, ignore this information altogether. As documents are

revisited, the link data, directory structure and document content information that is used by each scheme is also updated.

We highlight this process in Algorithm 5.1. The list L is initalised with the documents from a previous crawl, sorted using the ranking scheme being evaluated — PageRank for example. An initially empty list V is used to keep track of visited documents.

We then simulate the recrawl of the site by repeatedly removing the document from the front of list L and placing it in list V. Each time we remove a document d from list L we examine the outgoing links from the latest version of the document. Any documents that are linked to by the latest document d but are not in lists L or V are added to the front of list L. The order in which documents are removed from the front of the list becomes our crawl order. This process of removing documents continues until our cutoff value is reached.

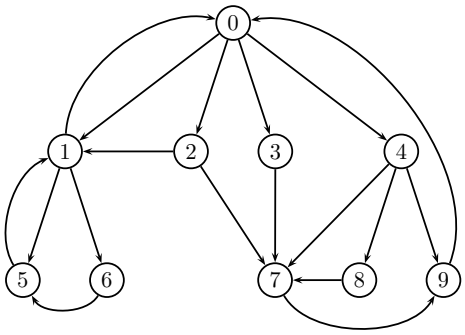
It is important to note that the list is ordered by the ranking scheme only once — before the crawl simulation begins. The only reordering that occurs after this is through the introduction of new documents, as just highlighted.

In the next section we highlight the various stateless crawl algorithms that we examine in our work.

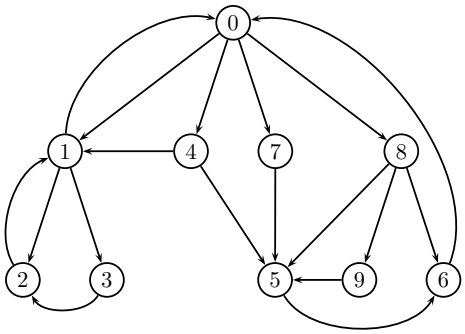
5.3 Crawler Strategies

In Section 2.2.2 (page 40) we highlighted the operation of several common crawl ordering schemes, namely the breadth-first, depth-first, and PageRank schemes. In this section we reintroduce these schemes and present several other crawl ordering schemes that we also compare in this chapter.

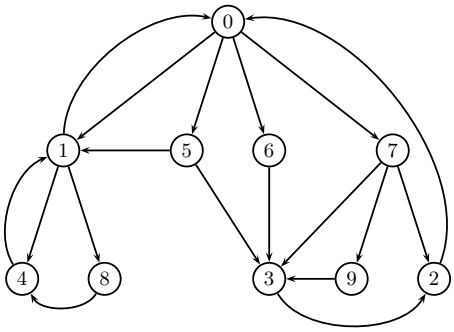
As we highlighted previously in Section 2.2 (page 30), a crawler starts at a seed document, visits the document, extracts all of the links from that document, and then subsequently follows each of the links to find more documents. We summarise this process with the simple algorithm in Algorithm 5.1. Step 7 ignores many of the problems in recognising mirrors and duplicate documents [Bharat and Broder, 1999], which we discussed in Section 2.13 (page 85). Here, we focus on Step 1; for crawling to be efficient, we must rank documents in L such that documents that have changed are more likely to be crawled than those that are static. This is analogous to ranking documents in response to a query, but rather than aiming for



(a) Breadth



(b) Depth



(c) PageRank

Figure 5.1: Crawl ordering schemes. The numbers in the nodes represent the order in which nodes are visited using each scheme, beginning at 0 and ending at 9. The arrows represent the HTML links between the nodes. In the case where several nodes are equally ranked, a breadth-first order among the nodes is used.

documents relevant to the query at the front of L, we want changed, popular documents at the front of L.

An effective method for predicting change is to look at past change statistics [Fetterly et al., 2003b; Malcolm and Armitage, 2003; Ntoulas et al., 2004a]. Documents that have changed frequently in the past typically change frequently in the future, possibly according to a Poisson model [Cho et al., 1998]. Such schemes are only effective if a long history of crawling has been established, and the statistics are stable over time [Cho and García-Molina, 2003a]. Here we consider queuing strategies that can be used without any prior knowledge of the change statistics of the web site that is the target of a crawl, or that can be used in conjunction with models based on change statistics.

Breadth

The BREADTH ordering scheme, as shown in Figure 5.1(a), orders a crawl by link depth, that is, the number of links that must be traversed from the root document to reach a particular document, retrieving documents that are located deeper in the link structure later. As each document is crawled, the URL links it contains are placed at the end of the queue. The BREADTH scheme penalises documents that are deeply nested, based on the principle that they require much deeper linear traversal to retrieve them from the root document, which in turn makes them less accessible to users.

It begins at the root document and explores all the neighbouring documents. Then, for each of these nearest documents, it explores their unexplored neighbouring documents, repeating the process until all documents are visited. The list, L is initialised with a breadth-first traversal of the link structure of a site beginning at the root document.

Breadth-first ordering has been shown to perform reasonably well in retrieving important resources early in a crawl [Cho et al., 1998; Najork and Wiener, 2001], particularly during the beginning of a crawl [Castillo, 2004].

Depth

The DEPTH ordering scheme, as shown in Figure 5.1(b), traverses the link structure, before retrieving documents in sibling branches. As each document is crawled, the URL links that

Algorithm 5.2: Breadth-first scheme.

Input: An input graph G of vertices V and edges E, the starting vertex s, array of vertices $out[u]$ linking out from vertex u

Variables: Vertices u and v, array $state[\,]$, integer i, queue Q, start of queue $head[Q]$, an empty array containing resulting ordered list $R[\,]$

Functions: ENQUEUE(Q, v) inserts vertex v into end of queue Q
DEQUEUE(Q) removes the vertex from start of queue Q

1: **foreach** *vertex* $u \in V[G] - \{s\}$ **do**
2: $state[u] \leftarrow$ UNSEEN
3: **end**
4: $state[s] \leftarrow$ SEEN
5: $i \leftarrow 0$
6: $Q \leftarrow \{s\}$
7: **while** $Q \neq 0$ **do**
8: $u \leftarrow head[Q]$
9: **foreach** $v \in out[u]$ **do**
10: **if** $state[v] =$ UNSEEN **then**
11: $state[v] \leftarrow$ SEEN
12: $R[i] \leftarrow u$
13: $i \leftarrow i + 1$
14: ENQUEUE(Q, v)
15: **end**
16: **end**
17: DEQUEUE(Q)
18: $state[u]$=visited
19: **end**

Algorithm 5.3: Depth-first scheme.

Input: An input graph G of vertices V and edges E, the starting vertex s, array of vertices $out[u]$ linking out from vertex u

Variables: Vertices u and v, array $state[\,]$, integer i, queue Q, start of queue $head[Q]$, an empty array containing resulting ordered list $R[\,]$

Functions: PUSH(Q, v) inserts vertex v onto top of stack Q
POP(Q) removes the vertex from top of stack Q

1: **foreach** *vertex* $u \in V[G] - \{s\}$ **do**
2: $state[u] \leftarrow$ UNSEEN
3: **end**
4: $state[s] \leftarrow$ SEEN
5: $i \leftarrow 0$
6: $Q \leftarrow \{s\}$
7: **while** $Q \neq 0$ **do**
8: $u \leftarrow head[Q]$
9: **foreach** $v \in out[u]$ **do**
10: **if** $state[v] =$ UNSEEN **then**
11: $state[v] \leftarrow$ SEEN
12: $R[i] \leftarrow u$
13: $i \leftarrow i + 1$
14: PUSH(Q, v)
15: **end**
16: **end**
17: POP(Q)
18: $state[u]$=visited
19: **end**

Algorithm 5.4: PageRank scheme.

Input: An input graph G of vertices V and edges E, number of iterations of the PageRank algorithm *iter*, constant dampening value σ, array of vertices $out[u]$ linking out from vertex u, array of vertices $in[v]$ linking into vertex v

Variables: Vertex u and v, array $Pagerank[\,]$

```
1:  foreach vertex u ∈ V[G] do
2:      Pagerank[u] ← 1
3:  end
4:  for i = 0 to iter do
5:      foreach vertex u ∈ V[G] do
6:          Pagerank[u] ← 0
7:          foreach v ∈ in[u] do
8:              Pagerank[u] ← Pagerank[u] + (Pagerank[v]/|out[v]|)
9:          end
10:         Pagerank[u] = (1 − σ) + σ · Pagerank[u]
11:     end
12: end
13: Sort Pagerank from highest to lowest Pagerank[u]
```

the document contains are placed at the front of the queue in the same way that a stack operates.

The DEPTH ordering scheme ranks documents in L by the order in which they occur in a depth-first traversal of the link graph of a site, and produces a more focused crawl, that retrieves all documents accessible via a particular link first, before moving onto the next link in a document.

In a closed web domain that is structured by topics, depth-first ordering would tend to gather documents on specific topics first. For this reason, past studies have used it as part of topic-driven crawling schemes [De Bra and Post, 1994].

Algorithm 5.5: Inlink scheme.

Input: An input graph G of vertices V and edges E, array of vertices $in[u]$ linking to vertex u

Variables: Vertex u, an empty array $N[\,]$

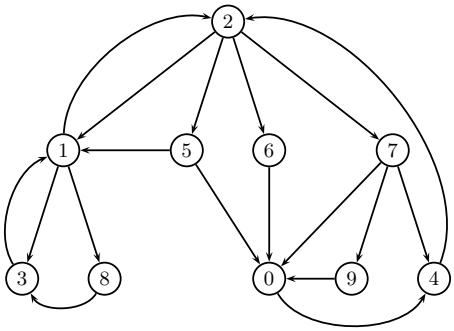
1: **foreach** *vertex* $u \in V|G|$ **do**
2: $N[u] \leftarrow |in[u]|$
3: **end**
4: Sort $N[u]$ from highest to lowest

PageRank

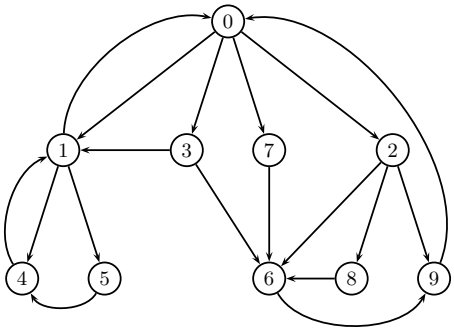
The PageRank ordering scheme, shown in Figure 5.1(c), and previously described in Section 2.2.1 (page 35), orders a crawl by the number of documents that link to a particular document and the number of documents that link to each of these documents. It also considers the number of outward links that each document has, such that the weight of each link is divided by the number of outward links the containing document has. Documents with high PageRank scores are considered important by web document authors, since many authors have created links to them. Furthermore, the recursive nature of the PageRank scheme means that links from documents with high PageRank have a greater weighting than links from documents with low PageRank. As we have highlighted in Section 2.2.1 (page 35), it is used to rank query results by search engines, such as Google [2008]. A study by Fagin et al. [2003a] on rank aggregation in an enterprise search environment has shown that the high-impact that PageRank has in improving ranking on the Internet is not replicated in an intranet environment. PageRank has also been compared to inlink with regard to predicting desirable documents and found to be similar in performance despite being computationally more expensive [Upstill et al., 2003a].

Inlink

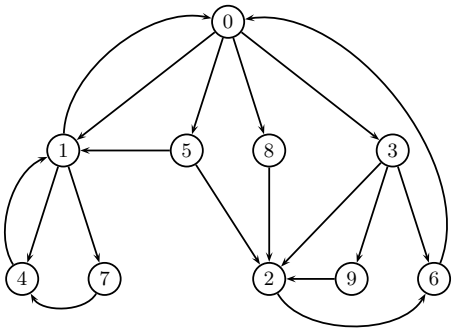
The Inlink ordering scheme, sometimes referred to as Backlink [Cho et al., 1998] or Indegree [Upstill et al., 2003b], determines the number of documents that link to a particular document and ranks documents in order from most highly linked to least linked, as shown in



(a) Inlink



(b) Outlink



(c) Hub

Figure 5.2: Crawl ordering schemes. The numbers in the nodes represent the order in which nodes are visited using each scheme, beginning at 0 and ending at 9. The arrows represent the HTML links between the nodes. In the case where several nodes are equally ranked, a breadth-first order among the nodes is used.

Algorithm 5.6: Outlink scheme.

Input: An input graph G of vertices V and edges E, array of vertices $out[u]$ linking out from vertex u

Variables: Vertex u, an empty array $N[\,]$

1: **foreach** *vertex* $u \in V|G|$ **do**
2: $N[u] \leftarrow |out[u]|$
3: **end**
4: Sort $N[u]$ from highest to lowest

Figure 5.2(a). That is, the more hyperlinks pointing to a document, the higher its perceived importance for crawling.

Previous studies with INLINK ordering have shown that its performance varies between moderate and poor when attempting to retrieve important documents earlier in a web crawl [Cho et al., 1998; Castillo, 2004]. A study by Fagin et al. [2003a] into rank aggregation in an intranet environment has shown that INLINK performs poorly when combined with traditional query-based ranking.

Outlink

The OUTLINK ordering scheme, sometimes referred to as Out-degree [Broder et al., 2000a; Henzinger, 2001; Dill et al., 2002] or Forward link [Page et al., 1998; Cho et al., 1998; Cho, 2001; Diligenti et al., 2004], determines the number of documents that a particular document links to and ranks documents in order from most outward links to least outward links, as shown in Figure 5.2(b). That is, the more outlinks a document contains, the higher its perceived importance for crawling. A document with many outlinks is likely to be a link collection, such as a sitemap, and thus is likely to have new links added to it periodically. A crawler may decide to frequently revisit a document with a high number of outlinks to ensure that the index contains as many of these new documents as possible.

A study by Pirolli et al. [1996] has shown that the OUTLINK scheme can be combined with other factors to identify index pages, that is, pages that serve as navigational link collections to pages that may or may not be related.

Algorithm 5.7: Hub scheme.

Input: An input graph G of vertices V and edges E, array of vertices $in[u]$ linking to vertex u, array of vertices $out[u]$ linking out from vertex u

Variables: Vertex u, an empty array $N[\,]$

1: **foreach** *vertex* $u \in V|G|$ **do**
2: $N[u] \leftarrow |in[u]| + |out[u]|$
3: **end**
4: Sort $N[u]$ from highest to lowest

Algorithm 5.8: URLdepth scheme.

Input: An input graph G of vertices V and edges E

Variables: Vertex u, an empty array $N[\,]$

Functions: URL(u) returns the URL of vertex u
SLASHES(U) returns number of slashes "/" in URL U

1: **foreach** *vertex* $u \in V|G|$ **do**
2: $N[u] \leftarrow$SLASHES(URL(u))
3: **end**
4: Sort $N[u]$ from lowest to highest

Hub

The HUB ordering scheme, as shown in Figure 5.2(c), ranks documents by the sum of the number of outlinks and inlinks in each document, combining the properties of the OUTLINK and INLINK schemes. The HUB scheme ranks highly documents that are link collections and are considered important by users. These documents are likely to be important hub documents, and hence, are good starting points for browsing users. An iterative variant of the HUB scheme called HITS, which we described in Section 2.2.1 (page 35), has been used in the past to order query results, using links graphs from the results subset [Kleinberg, 1999]. A weighted version of the HUB scheme has also been used in a study by Bharat and Henzinger [1998] to improve query results for given topics, producing comparable results to existing schemes, while reducing computation costs.

Algorithm 5.9: URLength scheme.

Input: An input graph G of vertices V and edges E

Variables: Vertex u, an empty array $N[\,]$

Functions: URL(u) returns the URL of vertex u

LENGTH(U) returns the length of URL U

1: **foreach** *vertex* $u \in V|G|$ **do**

2: $N[u] \leftarrow$LENGTH(URL(u))

3: **end**

4: Sort $N[u]$ from lowest to highest

URLdepth

The URLDEPTH ordering scheme determines the directory depth of a document in a domain, as opposed to the BREADTH scheme, which determines link depth. The URLDEPTH scheme favours documents that have a small depth over documents that are deeply nested. Documents are ordered by a score equal to the inverse of the number of slashes "/" in the URL of a document. URLDEPTH is based on the premise that documents with URLs that are located closer to the base document directory are generally more accessible and hence likely to be changing more frequently than documents that are deeply nested. Due to the general hierarchical tree-like structure of URLs, it is also reasonable to expect that the information located in deeper paths is more specific than the information located at smaller URL depths [Eiron and McCurley, 2003b]. Approaches similar to the URLDEPTH scheme have been used in the past for home-page finding tasks [Upstill et al., 2003b]. Home-page finding attempts to locate the entry page to a web site [Hawking and Craswell, 2002]. Past work has also demonstrated that URLs with fewer slashes are more useful [Arasu et al., 2001]. A previous study by Fagin et al. [2003a] has shown that the URLDEPTH scheme performs poorly at ranking query results in an enterprise environment when combined with traditional query-based ranking schemes.

Algorithm 5.10: Alpha scheme.

Input: An input graph G of vertices V and edges E

Variables: Vertex u, an empty array $N[\,]$

Functions: URL(u) returns the URL of vertex u

```
1: foreach vertex u ∈ V|G| do
2:     N[u] ←URL(u))
3: end
4: Sort N[u] alphabetically
```

URLlength

The URLLENGTH ordering scheme is very similar to the URLDEPTH scheme in many ways. It assumes that resources with short URLs are more likely to change frequently than long URLs. This is based on the assumption that short URLs are easier to remember, and are likely to be more accessible to users, and hence popular. Longer URLs are also more likely to contain slashes "/". It is easily computed by measuring the number of characters in each URL and ordering them from shortest to longest. The URLLENGTH scheme has been used in the past for ranking query results in intranet search in conjunction with traditional query-based ranking and found to perform well [Fagin et al., 2003a].

Alpha

The ALPHA scheme ranks documents based on the alphabetical ordering of their URLs and would tend to group resources by directory, since resources within the same directory would be alphabetically very similar. On a large domain, it is quite likely that resources in the same directory would have a similar content and frequency of change [Eiron and McCurley, 2003a]. Consider for instance, the "`news`" directory in the ABC domain, which could be expected to change fairly frequently.

Resources in the same directory would also tend to link to each other [Eiron and McCurley, 2003b;a], and so, when new resources are created, they would tend to be clustered in the same directory.

Algorithm 5.11: Query scheme.

Input: An input graph G of vertices V and edges E

Variables: Vertex u, an empty array $D[\,]$ and $N[\,]$, document d

Functions: INDEX(D) produce an index of a set of documents D

DOC(u) returns the content of the document in vertex u

QUERY(q, I, k) returns top k ranked documents for query q on index I

1: **foreach** *vertex* $u \in V|G|$ **do**
2: $D[u] \leftarrow$DOC(u)
3: **end**
4: $I =$INDEX(D)
5: **foreach** $q \in Q$ **do**
6: $R =$QUERY(q, I, k)
7: **foreach** $d \in R$ **do**
8: $N[d] \leftarrow N[d] + 1$
9: **end**
10: **end**
11: Sort $N[d]$ from highest to lowest

Query

The QUERY ordering scheme incorporates resource popularity into the crawling process. It ranks documents that are returned frequently to user queries, before documents that are seldom retrieved, hence attempting to update documents that have the most impact on search engine users. In our study we have evaluated resources from the previous crawl and considered them popular if they have been returned in the top thousand results fifteen or more times in response to about 18,000 queries collected from the ABC query logs and about 5,500 queries collected from CSIRO query logs.

While the QUERY scheme is similar to a focused crawl [Chakrabarti et al., 1999; 2002], in that it uses query topics to determine which resources to retrieve, there are several distinctions. In the QUERY scheme the query topics are not limited in scope, covering a wide range of distinct topics that are determined by their popularity. Furthermore, these topics are likely to change between crawls as their popularity changes over time [Diaz and Jones, 2004].

Large

The LARGE ordering scheme determines the number of documents in each directory including the documents in any of their sub directories, ranking documents from the largest directory branch first. If we assume that all directories begin with the same number of documents at the birth of a site, directories that are larger now must have had more new documents added to them than smaller directories. This scheme assumes that this trend will continue by ranking larger directories more highly than smaller directories. It also assumes that documents located in the same directory are more likely to be linked to each other [Eiron and McCurley, 2003b;a]. This scheme is similar to the larger sites first method [Baeza-Yates et al., 2005], but while that scheme orders a crawl by the size of a domain, the LARGE scheme orders by the size of a directory structure.

Random

The RANDOM ordering scheme reorders the crawl using random keys, and represents a baseline performance for comparing other schemes.

Algorithm 5.12: Large scheme insertion.

Input: An input graph G of vertices V and edges E, a starting vertex s, array of vertices $adj[c]$ adjacent to vertex c

Variables: Vertex u and c, an empty array $D[\,]$ and $N[\,]$, an array of tokens $T[U][\,]$ for URL U, document d, term t

Functions: URL(u) returns the URL of vertex u
TOKENISE(U, t) tokenises URL U on term t and return tokens as array
INSERT(t, c) creates new vertex for term t and adds edge to it from c
INC(t, c) increments frequency counter for term t adjacent to vertex c
TRAVERSE(t, c) traverses vertex containing term t adjacent to vertex c
INSERTURL(t, c) inserts the term t into the vertex c

```
1:  foreach vertex u ∈ V|G| do
2:      U ←URL(u)
3:      T[U] ←TOKENISE(U, '/')
4:      c ← s
5:      foreach term t ∈ T[U] do
6:          if t ∉ adj[c] then
7:              INSERT(t, c)
8:              INC(t, c)
9:              c ← TRAVERSE(t, c)
10:         else
11:             INC(t, c)
12:             c ← TRAVERSE(t, c)
13:         end
14:     end
15:     INSERTURL(U, c)
16: end
```

Algorithm 5.13: Large scheme.

Input: An input graph G of vertices V and edges E, a starting vertex s, array of vertices $adj[c]$ adjacent to vertex c

Variables: Vertex u and c, an empty array $D[\,]$ and $N[\,]$, an array of tokens $T[U][\,]$ for URL U, document d, term t

Functions: GetUrls(c) returns the URLs in vertex c
GetCount(c) returns the frequency counter for terms in vertex c
Traverse(t, c) traverses vertex containing term t adjacent to vertex c
Scan(c, n) recursively computes sum n of terms and vertices for vertex c

```
1: c ← s
2: n ← 0
3: Scan(c, n):
4:     n = n+GetCount(c)
5:     foreach term t ∈ adj[c] do
6:         D[n] ← GetURLs(c)
7:         c ← Traverse(t, c)
8:         Scan(c, n)
9:     end
10: end
11: Sort D[n] from lowest n to highest
```

Algorithm 5.14: Random scheme.

Input: An input graph G of vertices V and edges E

Variables: Vertex u, an empty array $N[\,]$

Functions: Rand(u) generates a random number using vertex u as a seed

```
1: foreach vertex u ∈ V|G| do
2:     N[u] ←Rand(u)
3: end
4: Sort N[u] from lowest to highest
```

	Document changed or new	Document not changed and old
Revisited	Popular – SUCCESS Not Popular – FAILURE	FAILURE
Not Revisited	Popular – FAILURE Not Popular – SUCCESS	SUCCESS

Table 5.1: Matrix for determining the success or failure of a crawl in acting on a single document that includes both freshness and popularity metrics.

5.4 Measuring Crawler Effectiveness

As we explained in Section 2.5 (page 48), crawler effectiveness is typically measured with a freshness metric that determines whether a document has been recrawled, and whether or not the document has changed or is new. However, this metric does not accurately reflect all the factors of the live Web. Most significantly, it does not consider the popularity of a document. Keeping popular documents current is essential, since these documents will be heavily utilised by end users, and changes made to these document are more likely to impact end users. Conversely, regularly updating documents that are rarely retrieved does not effectively utilise bandwidth of the search engine, and provides little benefit to end users. Accordingly, we extend the definition of success or failure of a crawler on a document, as in Table 5.1.

Popularity is determined by the queries issued by end users, hence it is more reflective of the end user's needs. In contrast, importance, typically measured using schemes such as PAGERANK and INLINK, indicates how highly linked to a document is from other documents, identifies how accessible the document is, and is determined by web authors. Although there is a correlation between importance and popularity, it is not unlikely for a document to have a high importance, as measured by PageRank and yet still have a low popularity, particularly if the topic is not queried frequently. For instance, a document containing copyright warning information is likely to be linked to by many documents in a domain, however, few users are

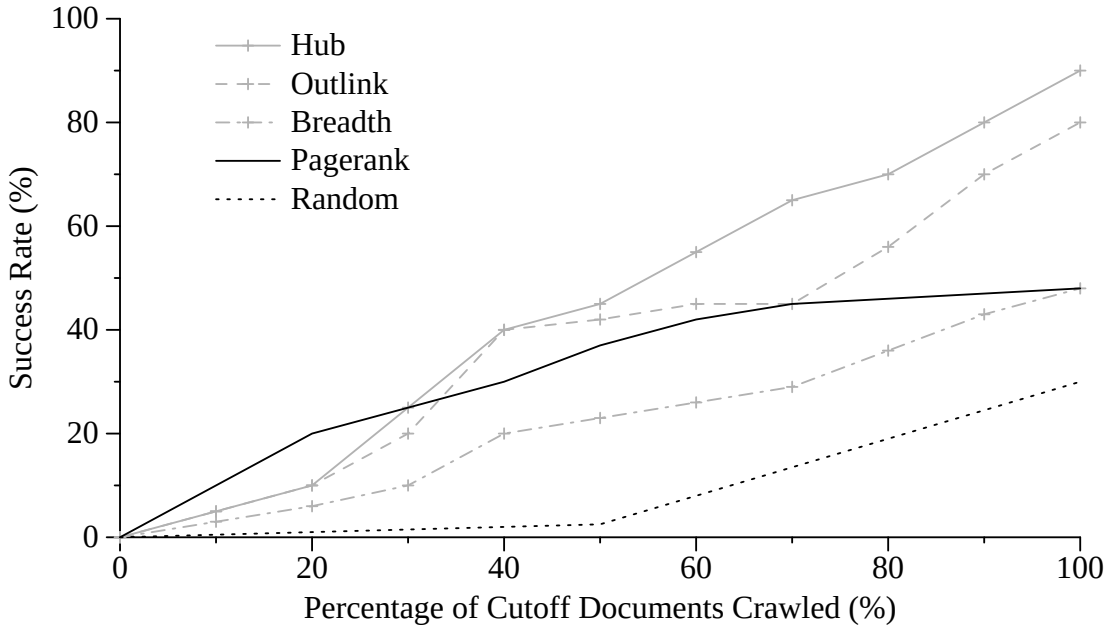


Figure 5.3: An example of the success rate of five schemes over a cutoff of 150,000 document retrievals on the ABC collection. The plots do not necessarily end at success rate of 100% since the crawl is restricted to 150,000 documents out of a total of approximately 540,000 (on average). Success rate *at any point in the crawl is defined as the number of changed or new documents retrieved divided by the number of documents retrieved at that point.*

likely to search for the copyright information in a domain, particularly since it is so readily available.

In our experiments, we determine popularity by computing the frequency with which a particular document is returned in response to user queries taken from the ABC and CSIRO query logs. Since we used the single source of queries to determine popularity for both the QUERY ordered scheme as well as evaluating all the schemes, we required a way to avoid skewing the results.

To avoid this problem, we acquired approximately eight weeks of ABC queries and eleven weeks of CSIRO queries, then, for each domain, we took their respective queries, sorted them by date, and split each week of queries into two equal sequential sets of queries. The first half was used for producing the QUERY ordered crawl, while the second half was used to evaluate the schemes. This gives us about 18,000 queries for evaluating the ABC collection and about 5,500 queries for evaluating the CSIRO collection.

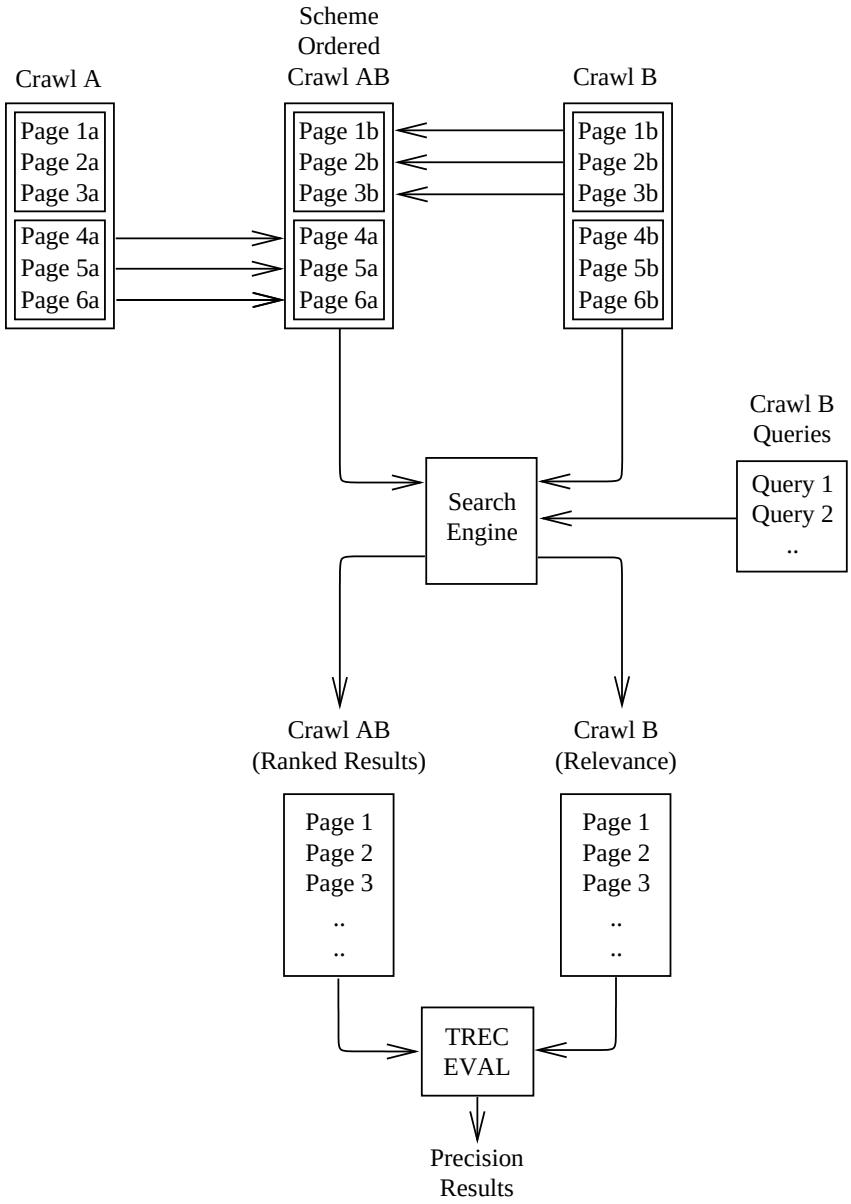


Figure 5.4: Crawl ordering evaluation. The scheme crawls all documents during Crawl A and then decides on which subset to recrawl during Crawl B. This produces Crawl AB consisting of a subset of documents from Crawl A and from Crawl B. Documents from Crawl B are guaranteed to be fresh, while documents from Crawl A may not be up-to-date if they have changed during crawl B. User queries are used to rank and compare Crawl AB and Crawl B documents. Crawl AB becomes the scheme being evaluated, while Crawl B becomes the relevance judgements.

As we observed in Section 2.2.2 (page 40), each scheme is a ranking of documents, just like in the operation of a ranking algorithm in a search engine. Here, however, rather than ranking documents by their likelihood of relevance to a query, as in a search engine, we are ranking documents by their likelihood to have changed since the last crawl. Furthermore, rather than using relevance to judge a ranking of the documents, we are using the standard success measures we described in Section 2.5 (page 48) and Table 2.1 (page 50). In addition, we take popularity into consideration by using the measures shown in Table 5.1. Another difference from the usual retrieval problem is that new documents, not included in the original ranking of documents, are always a success under the standard matrix, and might be a success under the popularity matrix.

We define the *success rate* at any point in the crawl to be the number of successes achieved divided by the number of documents crawled. The *final success rate* is the success rate at the end of the crawl. The *average success rate* is the average of all success rates achieved after every document crawled. For example, consider the graphs of success rate shown in Figure 5.3. The final success rate of the PageRank and Breadth schemes is 48%, but the average success rate of PageRank is 35%, while Breadth has an average success rate of about 25%.

5.5 Measuring Impact on Users

In order to determine the impact that crawl algorithms have on user search results, we also evaluate crawls using an alternative metric that takes popularity into consideration. This technique also uses queries as part of the evaluation and is very similar to the technique we used in Section 4.4 (page 108). We illustrate the method in Figure 5.4, and describe it below.

The technique begins by crawling all documents during the first crawl — Crawl A in Figure 5.4. We then use the crawl ordering scheme to decide which documents to revisit during the second crawl — Crawl B in Figure 5.4. This results in Crawl AB, a hybrid collection that consists partially of the up-to-date collection from Crawl B and the stale collection from Crawl A. This collection is compared to the fully up-to-date collection, Crawl B, by running a set of queries against the partially up-to-date collection and the fully up-to-date collection. The search results of these two collections are compared using the TREC evaluation system to determine how document rankings have been affected. If the crawl scheme is effective,

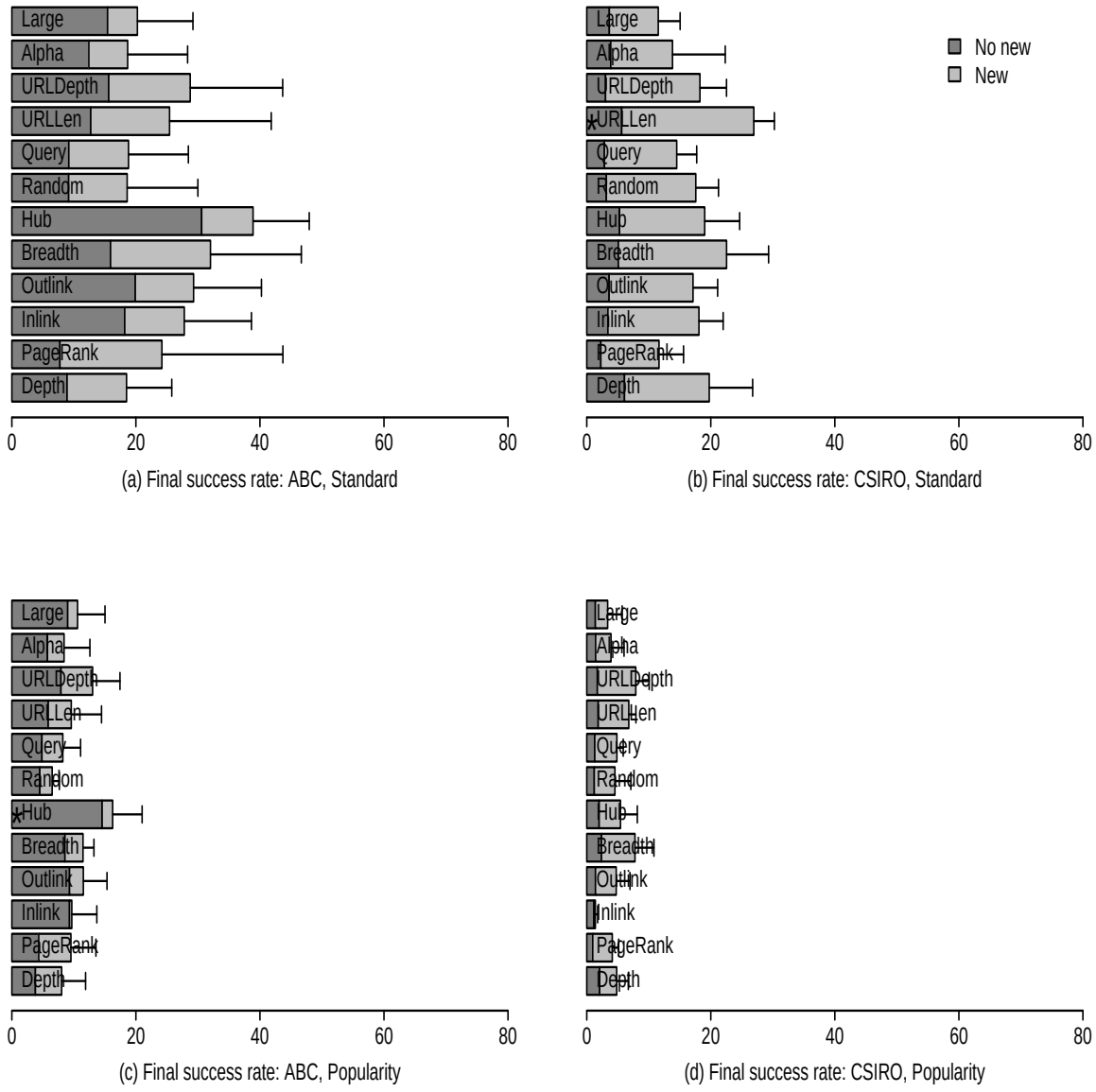


Figure 5.5: Final success rate (%) over all documents, averaged over all intervals. Error bars indicate one standard deviation. Light bars include new documents, contrary to dark bars which ignore new documents. Stars indicate a significance to Random method (Tukey HSD test, $p < 0.05$).

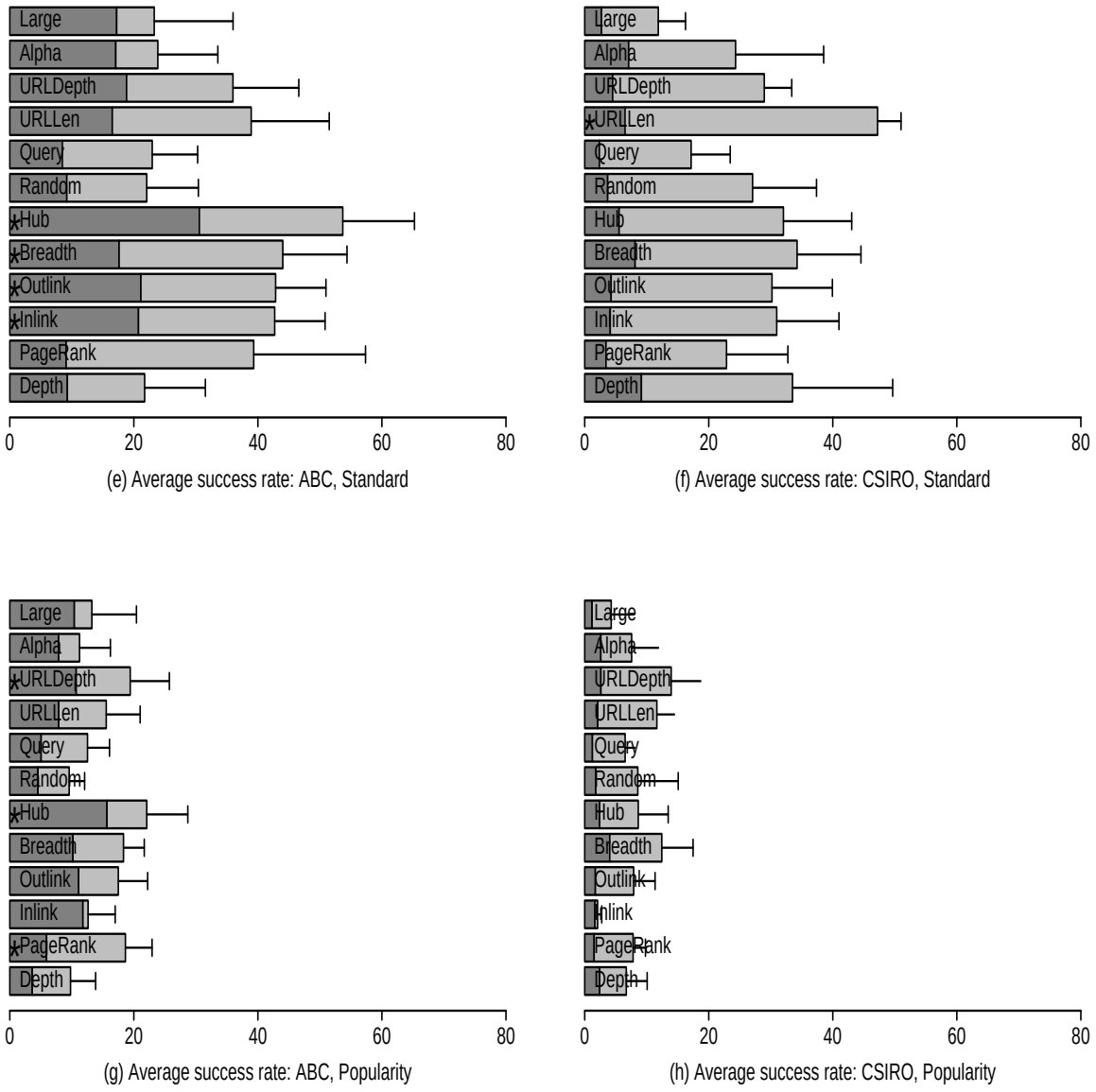


Figure 5.6: Average success rate (%) over all documents, averaged over all intervals. Error bars indicate one standard deviation. Light bars include new documents, contrary to dark bars which ignore new documents. Stars indicate a significance to Random method (Tukey HSD test, $p < 0.05$).

then there should be no changes in the rankings. While documents may have changed, if the rankings have not, we assume that this means that the document is still relevant to the query. Furthermore, by using user queries, we are ensuring that a greater emphasis is placed on retrieving popular documents rather than unpopular documents that are not returned as query results.

In the next section we discuss the various results we achieved throughout our experiments into stateless crawl ordering.

5.6 Results

In this section we present the results of our crawl experiments on the ABC and CSIRO collections, using the various crawl methods outlined in Section 2.2.2 (page 40) under various test conditions incorporating popularity and change.

Figures 5.5 and 5.6 show the mean average precision (MAP) of each recrawl scheme, broken down into the two components *No new* and *New*. The *No new* component consists of documents that were crawled in a previous interval, and have been found to have changed during a subsequent crawl, while the *New* component consists of documents that had not been discovered in any of the previous crawls of the domain. Note that we are using the full history of the collection to *evaluate* the schemes, but we are only using the previous interval as *input* to any one of the schemes, thus ensuring that the schemes operate in a stateless fashion.

In Figures 5.5(a-b) and 5.6(e-f) document popularity is ignored, and only documents that change by more than the WORDS threshold are considered a success. In Figures 5.5(c-d) and 5.6(g-h) document popularity is included in the definition of success. Furthermore, while all documents are retrieved during the first crawl of a site, only 150,000 documents are retrieved on subsequent crawls of the ABC domain, and 45,000 of the CSIRO domain; thus simulating a production crawler that cannot afford bandwidth to recrawl the entire site every crawl.

Using this criteria, the best performance on the ABC when ignoring popularity, Figures 5.5(a) and 5.6(e), is the HUB method, although using the final success rate no method is statistically significantly better than RANDOM. Using the average success rate method, the OUTLINK, INLINK, and BREADTH schemes outperform RANDOM. Comparing panel (a) with

panel (e), it is apparent that new documents are the main contribution to the average success rate succeeding the final success rate (the light section of the bars are longer). This is because new documents are discovered earlier in the crawl, rather than later. The success rate early in some crawls is up around 80%, due to new document discovery, falling eventually to the final rate that is less than 40% (on average). The same pattern is evident when comparing Figures 5.5(b) and 5.6(f).

For the CSIRO web site, when popularity is ignored, the URLLENGTH scheme outperforms all others. In fact, it is the only scheme that is superior to RANDOM.

In Figures 5.5(c-d) and 5.6(g-h), document popularity is taken into consideration, reducing the overall number of documents that are considered successes, since a successful document must be both changed and popular. The majority of schemes have a success rates of about half of the previous standard value. From the last row of Tables 3.2 (page 97) and 3.4 (page 98) we can see that about half of all changed documents are popular, so we would expect the success rate for finding changed documents to drop by about 50%, on average, when popularity is taken into account. Comparing the dark bars in Figure 5.5(a) and (c), for example, shows that this is the case. With popularity taken into account, HUB remains the dominant scheme on the ABC site, but no method stands out on the CSIRO site.

Surprisingly, the introduction of the popularity success criteria did not boost the performance of the QUERY scheme. In separate experiments not reported here, we found that when we used the same queries for both ordering L in the QUERY scheme, and for evaluation of success, QUERY still did not perform well.

It is also worth noting that the majority of successess in the CSIRO crawls were derived from the discovery of new documents: that is, the light bars dominate the dark, unlike in the ABC crawls. From Tables 3.2 (page 97) and 3.4 (page 98), we can see that the number of new documents in each CSIRO crawl were about two to four times more than the number of changed documents, while in the ABC crawls, the number of new documents was usually less than the number of changed documents. We have since learned that the CSIRO domain was undergoing major restructuring around the time of the crawl period, leading to the volatility of the site and the creation of many new documents.

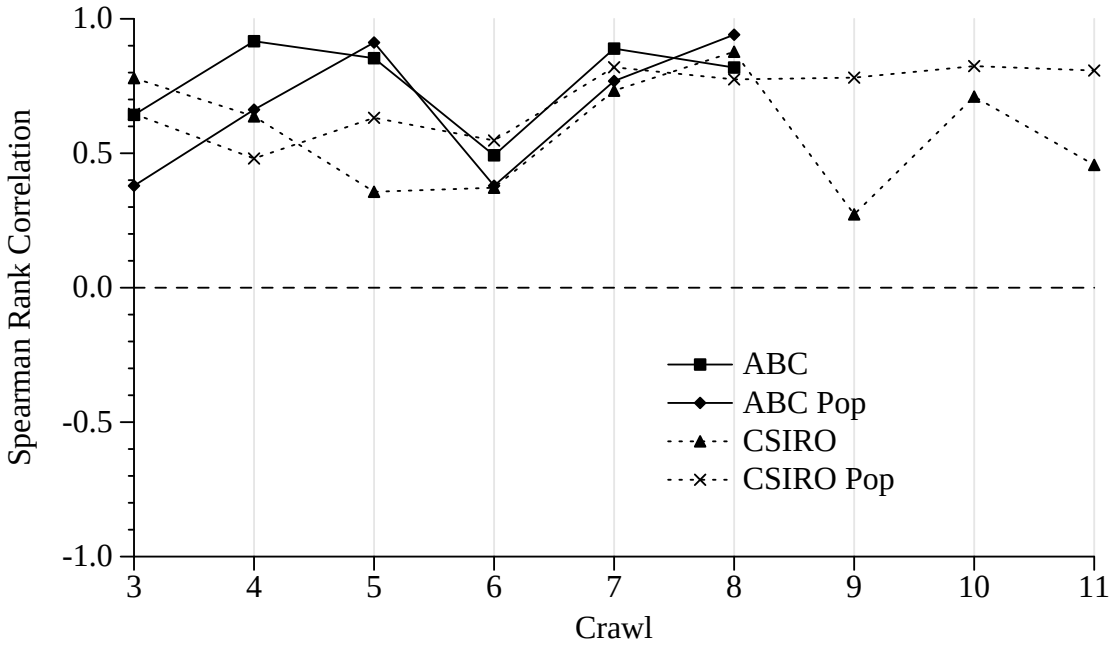


Figure 5.7: Correlation between the rankings of schemes by average success rate across all crawls in our test suite.

Query	Hub	PageRank	Breadth	URLdepth	Random	Inlink
88.35	78.11	69.21	67.12	62.11	61.91	59.40

Table 5.2: Effectiveness of top seven schemes averaged over seven recrawls of the ABC test collection. Schemes are ordered from most effective to least.

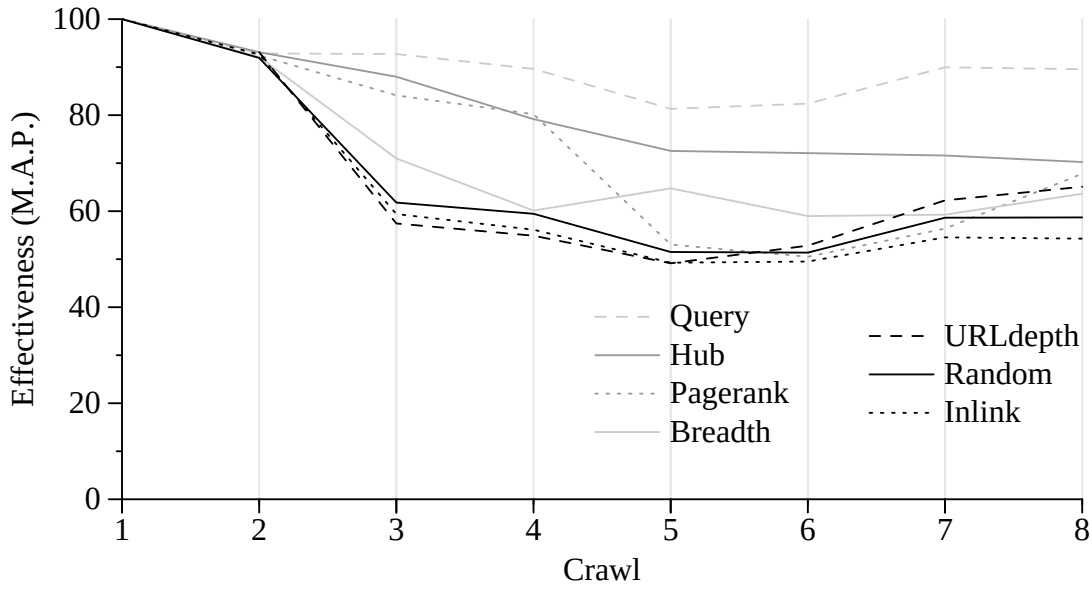


Figure 5.8: Performance of schemes during each crawl on the ABC collection using the alternative evaluation technique outlined in Section 5.5 and Figure 5.4. Effectiveness is measured using Mean Average Precision between the query results on the up-to-date collection and the partially updated collection on each crawl. During each crawl only 150,000 additional documents are updated.

The result from the next analysis is presented in Figure 5.7; a plot of the Spearman Rank Correlation Coefficient between a ranking of the schemes at each interval. Along the x-axis the current interval is shown, and at each interval the mean of all previous ranks for a scheme are used to evaluate the relative performance of each scheme. That is, the more intervals that are used, the more past statistics that are used to predict the future performance of a scheme. From the graph it is clear that there is a strong correlation between the relative performance of different schemes as the number of intervals increases. This means that after very few intervals the relative performance of different schemes becomes apparent and more importantly the performance is fairly consistent. Hence once a good scheme for the domain is determined, it remains a good scheme.

In the next set of results, we examine the performance of the top seven schemes over each crawl of the ABC collection. We use the alternative evaluation technique highlighted in Figure 5.4 (page 150) and Section 5.5 (page 151) to evaluate their performance. During the

first crawl all documents were retrieved. We determined the effectiveness of each scheme by limiting subsequent crawls to 150,000 documents. We then ran queries against each partially updated collection to produce a set of ranked results. The same queries were run against a fully up-to-date collection. We then determined the similarity between these results using Mean Average Precision.

Table 5.2 highlights the performance of the schemes averaged over seven recrawls of the ABC collection, while in Figure 5.8 we break down the performance during each crawl. The x-axis in Figure 5.8, represents the crawl number being evaluated; at each crawl another 150,000 documents are updated in the collection. The y-axis is the effectiveness of each scheme.

Although we tested all schemes using this evaluation technique we only report the schemes that are on average at least as effective as the random approach.

On the first crawl all documents are retrieved and so the effectiveness for all schemes is 100%. On the second crawl 150,000 documents are updated, however each scheme performs almost identically. On the third crawl, there is a much greater variance. The effectiveness of the Breadth, Random, Alpha, URLdepth, and Inlink schemes all drop sharply below 71%, while the effectiveness of the Hub and PageRank schemes drops slightly to 88% and 84% respectively. Only the performance of the Query scheme remains steady at 93%. On the fourth crawl the performance of all schemes drops, though to varying degrees. The top three schemes, Query, Hub, and PageRank respectively, all have an effectiveness that is 20%-30% higher than that of the Breadth, Random, URLdepth, and Inlink schemes. Interestingly, the effectiveness of the PageRank scheme drops sharply to roughly that of the Random scheme, from crawl five onwards. Throughout all crawls of the ABC the Query scheme outperforms all others, maintaining an average effectiveness of roughly 88% — over 10% more effective than the Hub scheme.

Our results clearly show that if the relevance of user query results is to be maintained, a scheme must incorporate query terms. Our observations in Tables 3.2 and 3.4 (page 97 and 98), demonstrated that the percentage of documents that were changed and returned by queries was clearly a small subset. By focussing a crawl on these documents the greatest impact on search results can be made.

URLlength	Breadth	Depth	Query	URLdepth	Inlink	Random
86.93	85.87	85.72	85.68	85.16	83.40	82.92

Table 5.3: Effectiveness of top seven schemes averaged over ten recrawls of the CSIRO test collection. Schemes are ordered from most effective to least.

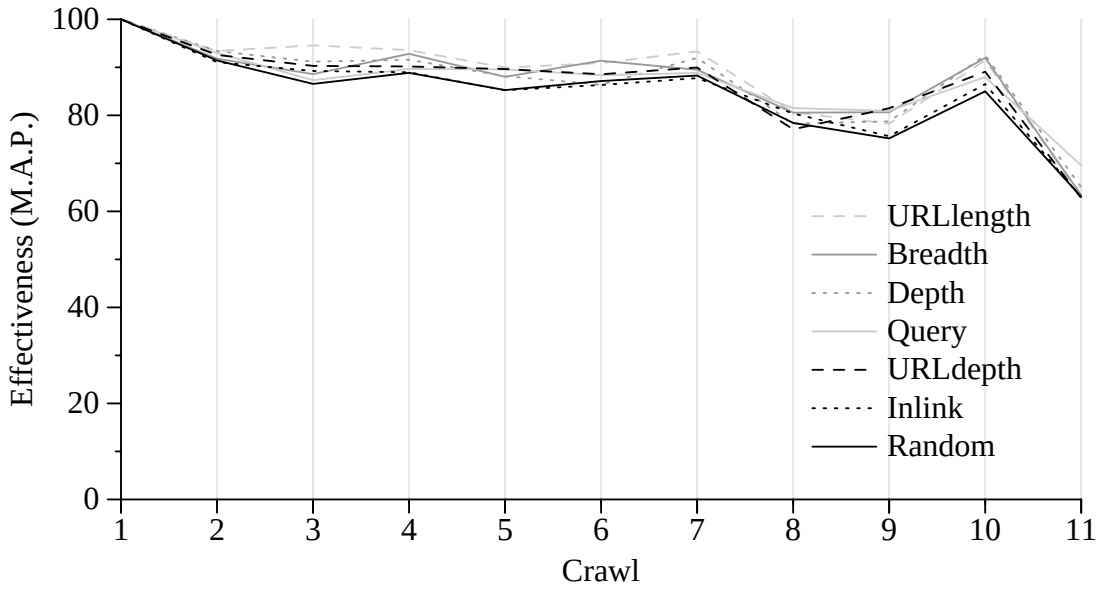


Figure 5.9: Performance of schemes during each crawl on the CSIRO collection using the alternative evaluation technique outlined in Section 5.5 and Figure 5.4. Effectiveness is measured using Mean Average Precision between the query results on the up-to-date collection and the partially updated collection on each crawl. During each crawl only 45,000 additional documents are updated.

We also used the same evaluation techniques on the CSIRO collection, presented in Table 5.3 and Figure 5.9. Our results indicate that all schemes performed equally well. When we tested an approach that did not update any documents during each interval, we found that the results were still quite high — with an average effectiveness of over 78% — compared to 86% for the URLLENGTH scheme, and 82.92% for the RANDOM scheme. This indicated that performance similarity was not caused by our 45,000 document cutoff being too high. Suspecting that the number of relevant documents per query was low, we then examined the collection. We found that the number of relevant documents per query was 25% fewer than for the ABC collection.

Based on this result, and the observed volatility of the collection due to restructuring, we felt that the CSIRO collection would not be useful for comparing schemes. While we continue to use the CSIRO collection in further work, we report primarily on our results for the ABC collection.

Next, we analyse the complexity of the various schemes that we have implemented in this Chapter.

5.7 Analysis

In this section we examine the costs and savings of the crawl schemes highlighted in this chapter. Specifically, we measure and compare the computational costs and bandwidth savings of each scheme. We assume that a collection of N URLs need to be sorted according to a crawl algorithm.

In the case of the BREADTH, DEPTH, and RANDOM schemes, the URL of each page is read and then written into a queue with no other processing required. Hence the cost for these schemes is given as

$$\begin{aligned} Cost &= N + N \\ Cost &= O(N) \end{aligned} \tag{5.1}$$

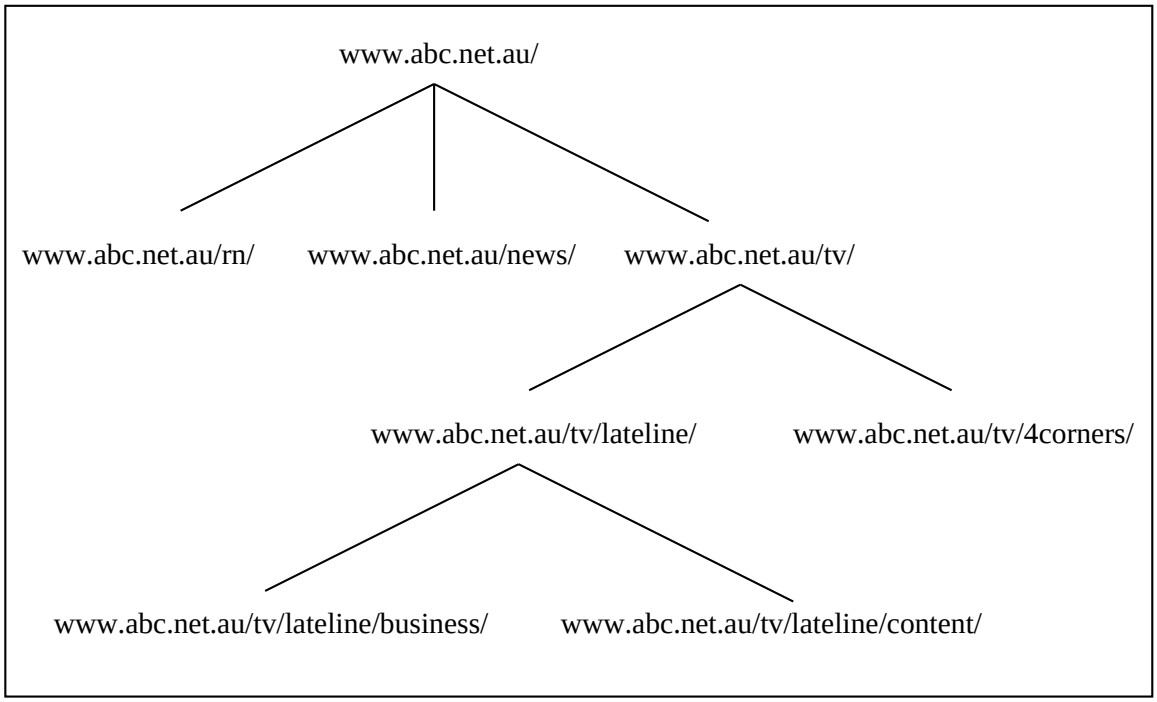


Figure 5.10: URL tree structure.

The URLLENGTH, ALPHA, and URLDEPTH schemes require only a URL sort based on the relevant key, and so, the cost for these schemes reduces to the cost of the sort.

$$Cost = N \log N$$
$$Cost = O(N \log N) \tag{5.2}$$

The INLINK scheme determines the number of pages that link to each given page, and so, the scheme must retrieve the list of inward links for each URL, then sort the URLs by the number of links.

$$Cost = (N \times I) + N \log N$$
$$Cost = O(N \log N) \tag{5.3}$$

where I is the average number of links to each page.

The OUTLINK scheme is similar to the INLINK scheme, except that instead of examining the number of inward links, it determines the number of outward links each page has, and so, the cost is

$$Cost = (N \times O) + N \log N$$
$$Cost = O(N \log N) \tag{5.4}$$

where O is the average number of links originating from each page.

The PageRank scheme must determine the pages that link to, and that are linked by, each page. In addition, there must be several iterations before the initial scores of each page converge with their true values. Finally, pages must be sorted by their respective scores.

$$Cost = (((N \times I) + (N \times O)) \times E) + N \log N \tag{5.5}$$

where E is the number of iterations of the algorithm.

The Hub scheme must determine the number of inlinks and outlinks that each page has, these values are summed together and pages are then sorted accordingly.

$$Cost = (N \times I) + (N \times O) + N \log N \tag{5.6}$$

The Large scheme must determine the number of URLs located in each directory and any sub-directories. The URL directory structure must be represented as a tree as shown in Figure 5.10. Creating this structure has the cost

$$Cost = N \times T \tag{5.7}$$

where T is the average number of slashes "/" in a URL.

Once the tree is created, the number of sub-directories must be determined for each directory. If the process begins at the root node (www.abc.net.au/), and we assume that past statistics regarding sub-directories are ignored, all sub-directories must be re-examined for each directory, and so, the cost is given as

$$Cost = H \times N' + H - \sum_{i=1}^{H}(A^i)$$

$$Since \sum_{i=1}^{H} A^i = \frac{A^{H+1}-1}{A-1} - 1$$

$$Cost = H \times N' + H - \frac{A^{H+1}-1}{A-1} - 1$$

$$Given\ H \approx \lceil \log_A N' \rceil$$

$$Cost = \lceil \log_A N' \rceil \times N' + \lceil \log_A N' \rceil - \left(\frac{A^{\lceil \log_A N' \rceil + 1} - 1}{A-1} - 1 \right)$$

$$Cost = \lceil \log_A N' \rceil \times (N'+1) - \left(\frac{A \times A^{\lceil \log_A N' \rceil} - 1}{A-1} - 1 \right)$$

$$Cost = \lceil \log_A N' \rceil \times (N'+1) - \left(\frac{A \times N' - 1}{A-1} - 1 \right)$$

$$Cost = \lceil \log_A N' \rceil \times (N'+1) - \left(\frac{A \times N' - 1}{A-1} \right) + 1$$

$$\implies Cost = O(N' \log_A N') \tag{5.8}$$

where N' is the number of unique directories, H is the average height of the tree, and A is the average number of immediate subdirectories in each directory. In the worst case the directory structure is deep (H is large) and the directories contain few branches (A is small) while there are many distinct directories (N'). In the worst case the computation cost is

$$Cost = \frac{N'(N'-1)}{2} = O(N'^2) \tag{5.9}$$

An improvement to the LARGE scheme is to work from the bottom of the tree to the top, thus avoiding recalculating the subdirectories contained in each directory. The computation cost then becomes

$$Cost = (N' \times A) \tag{5.10}$$

Hence, the final cost of creating the tree, determining the number of nodes and sub nodes, and sorting the URLs accordingly is

$$Cost = (N' \times A) + (N \times T) + (N \log N) \tag{5.11}$$

The QUERY scheme determines the frequency that documents are returned in response to queries. This requires documents to be indexed, running the queries and noting the top ranked documents, then finally sorting the crawl accordingly.

$$\begin{aligned} Index\ Cost &= \frac{N^2 \log T}{2B} \\ Cost &= \frac{N^2 \log T}{2B} + C \times (Q + K) + N \log N \\ Cost &\approx N^2 \end{aligned} \tag{5.12}$$

where T is the number of distinct terms in the indexed collection, B is the number of postings in memory, C is number of queries run (18,000 in our work), Q is the cost of running a single query (very small), and K is the maximum number of documents returned in response to a query (1000 in our work).

5.8 Summary

In this chapter we have investigated strategies for crawl ordering that aim to recrawl documents that have changed and avoid recrawling static documents. The schemes we have examined are intended for cases where no past change statistics are available, as is the case when crawling a new domain, or when discovering new documents in an existing domain. We have used existing schemes from the literature, and introduced the QUERY schemes and our simplified implementation of the HUB scheme. We have also introduced the notion of popularity in evaluating the crawls.

On the ABC domain, the HUB scheme had the highest overall success rate over all intervals, and was statistically significantly better than a randomly ordered crawl. This was true using both the criteria for success that a document had changed, and the more stringent criteria that a document had changed and was popular. On the CSIRO domain, the URLLENGTH scheme was the best performer when change was the measure of success, but when popularity was also required, no method outperformed a randomly ordered crawl. This demonstrates what is perhaps intuitive: different domains may be better served by different crawl strategies. It also highlighted a limitation of our CSIRO collection.

We have also demonstrated that the performance of schemes over a sequence of crawls on the same domain is relatively stable. The correlation between the ranking of schemes

from one interval to the next (as shown in Figure 5.7) is high: generally around 80% to 90%. Not only are different schemes suitable for different domains, but, once a good scheme has been established, it remains good for future crawls. Again this supports our intuitive understanding of web sites within an enterprise. They are constructed by organisations with specific goals in mind, and they are maintained according to regular policies and rules. Hence once the appropriate crawling scheme is chosen within an enterprise, it continues to apply while the organisation maintains its site according to those rules.

In this chapter we have introduced the QUERY scheme for ordering documents for recrawl based on past query patterns. The scheme did not work any better than randomly selecting documents during a crawl on either site (Figures 5.5 and 5.6). While we had hypothesised that biasing a crawl by previous queries would help the crawler find popular documents, the impact on changed and new documents was uncertain. If a large proportion of successes in the crawls were due to the discovery of new documents — as in the case of the CSIRO collection — the QUERY scheme may be expected to perform poorly. Past studies have shown, for example, that the PageRank scheme is biased against new documents [Baeza-Yates and Castillo, 2001; Baeza-Yates et al., 2004], since users are less likely to link to newly created documents. In a similar way, users may be less likely to query these same documents.

When we examined the performance of various schemes by evaluating their impact on search results, we found that while the HUB scheme still performed well for the ABC collection, the QUERY scheme outperformed all other schemes. This highlights the need for the incorporation of query statistics into the crawling process if user query results are to be maintained effectively.

Overall however, our results suggest that the distribution of popular documents is essentially random from a change perspective. Users are interested in a wide variety of documents, and this is not necessarily tied into their likelihood of changing. This does not however dismiss the fact that documents that are popular cannot be ignored. It means that change and popularity are two possibly competing factors, and so, a crawl scheme must compromise between these two objectives. In the next chapter we introduce a scheme that allows these factors to be considered as part of a general crawling scheme while maintaining crawl effectiveness and efficiency.

Chapter 6

Change-Driven Crawling Using Anchor Text

Web crawlers need to revisit documents periodically to ensure that query results are relevant to user queries. Typically, schemes that predict change frequency either require past change statistics, use statistics that are susceptible to tampering, or assume that the documents change according to some particular model.

We introduce a novel scheme for ordering the crawl frontier by examining the anchor text pointing to each document. Our scheme begins with an ordering based on the previous crawl, but adapts during the current crawl as anchor text term statistics change. Section 6.1 explains the method in more detail. We test our scheme on snapshots of two web sites, and in Section 6.3, we show that it is more efficient than previous schemes proposed in the literature.

Instead, our scheme only requires one complete crawl. Then on the second crawl, as each document is retrieved, the scheme updates the change statistics of each anchor term pointing to the retrieved document. If the retrieved document changed, the score *increases* for each anchor term that points to the document. If the retrieved document was unchanged the score *decreases* for each term. As the crawl continues, the changing term scores affect which document will be crawled next. In this way, the crawler adapts to actual document change statistics as they are detected dynamically during the crawl. While document content could be used instead of anchor text to predict change, this would be much more computationally

Algorithm 6.1: Simple crawl scheme.

Input: List L of all documents d from a previous crawl, an empty list V

```
1: Sort L by some ranking scheme
2: while L is not empty and bandwidth is available do
3:     Visit the first document d in list L
4:     Remove d from L and insert into V
5:     S ← documents linked to by d
6:     foreach s ∈ S do
7:         if s is not in L or V then
8:             Add s to the front of L
9:         end
10:    end
11: end
```

expensive and may not necessarily improve results, as anchor terms have been shown to be quite effective in identifying a document [Craswell et al., 2001; Hawking, 2004] or predicting the target document's content [Amitay, 1998].

6.1 Dynamic Crawl Adaptation

In this section we describe our dynamic crawl adaptation scheme in detail. During the initial crawl *all* documents are crawled using the method highlighted in Algorithm 6.1. Since all documents are crawled, this provides our scheme with the complete link graph structure and *anchor terms* statistics required by the schemes.

Further to this, we define:

- d , a web document;
- A_d, the set of stemmed and stopped anchor terms that point to d;
- c_t, the number of times anchor term t points to a changed document;
- u_t, the number of times t points to an unchanged document; and
- f_{td}, the number of times anchor term t points to document d.

Algorithm 6.2: Dynamic crawl adaptation scheme.

Input: An input graph G of vertices V and edges E, integer *limit* number of documents to recrawl during each crawl, array of vertices $out[u]$ linking out from vertex u, array of edges $inEdge[v]$ linking into vertex v

Variables: Vertices u, v, and $RevisitPage$, array $state[\,]$, integers i and j, doubles $MaxScore$ and $Score$, terms s and t, arrays $anchors[\,]$ and $CheckAnchors[\,]$

Functions: CalculateScore(u) returns the change score for vector u
Retrieve(u) recrawls vertex u
GetAnchors(E) returns list of all anchors that are in the set of edges E
AlterChangeStats(v) updates change/unchange statistics for vector v

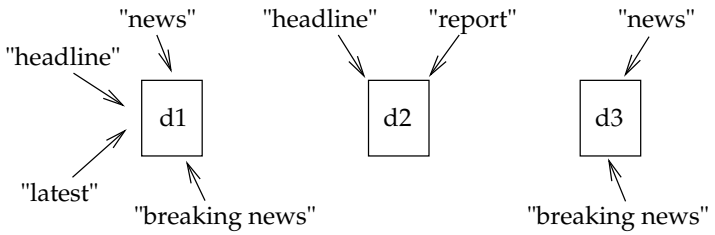


Figure 6.1: An example site with three documents, and the anchor terms pointing to them.

The score of document d is then calculated as the average change-to-unchange ratio over all terms pointing to the document:

$$S(d) = \frac{1}{|A_d|} \sum_{t \in A_d} \frac{c_t}{c_t + u_t} \tag{6.1}$$

To take into consideration the frequency with which a particular term points to a document, we use a modified score:

$$S_f(d) = \frac{1}{|A_d|} \sum_{t \in A_d} \left(\frac{c_t}{c_t + u_t} \times f_{td} \right) \tag{6.2}$$

Hence, the more frequently a term is included in anchor text that points to a document, the more weight it has on the documents score when using S_f.

The steps in the crawling process are shown in Algorithm 6.2 and 6.3. We outline these steps through the use of a sample site containing three documents with their associated anchor links, shown in Figure 6.1.

Algorithm 6.3: Dynamic crawl adaptation scheme (cont.).

```
1  foreach vertex u ∈ V[G] do
2      state[u] ← UNVISITED
3  end
4  for j ← 0 to limit do
5      MaxScore ← 0
6      RevisitPage ←GetDefault(G)
7      foreach vertex u ∈ V[G] do
8          if state[u] = UNVISITED then
9              Score=CalculateScore(u)
10             if Score > MaxScore then
11                 MaxScore ← Score
12                 RevisitPage ← u
13             end
14         end
15         Retrieve(RevisitPage)
16         state[RevisitPage] ← VISITED
17         anchors[ ] ←GetAnchors(inEdge[RevisitPage])
18         foreach term t ∈ anchors[ ] do
19             foreach vertex v ∈ V[G] do
20                 CheckAnchors[ ] ←GetAnchors(inEdge[v])
21                 foreach term s ∈ CheckAnchors[ ] do
22                     if s = t then
23                         AlterChangeStats(v)
24                     end
25                 end
26             end
27         end
28     end
29 end
```

	Term				
	"news"	"headline"	"latest"	"breaking news"	"report"
Before d1					
Change Freq (c_t)	0	0	0	0	0
Unchange Freq (u_t)	0	0	0	0	0
After d1					
Change Freq (c_t)	1	1	1	1	0
Unchange Freq (u_t)	0	0	0	0	0

Table 6.1: Change statistics for the anchor terms pointing to the documents in example in Figure 6.1. The first set of term change and unchange frequency statistics are produced before any documents have been retrieved. The second set are produced after document one has been retrieved. In this case document one has changed.

In Table 6.1, the change statistics for each term are shown. Since no documents have been visited yet, all terms initially have a change and unchange frequency of zero and hence all documents have the same score. In such cases, the default ordering is used, such as PageRank [Page et al., 1998], or breadth-first ordering. In this case assume that *document one* is retrieved first and has changed. Accordingly, each term pointing to *document one* has its change frequency (c_t) incremented, while its unchange frequency (u_t) remains unchanged, as shown in Section 2 of Table 6.1. The next step is to calculate the score of the two remaining unvisited documents, *documents two* and *three*. The score for *document two* is calculated by averaging the score of the terms "headline" and "report", while the score for *document three* is the average score of the terms "news" and "breaking news".

$$d2 = \left(\frac{1 + 0.00001}{0 + 0.00001} + \frac{0 + 0.00001}{0 + 0.00001}\right) \div 2 = 50,001 \tag{6.3}$$

$$d3 = \left(\frac{1 + 0.00001}{0 + 0.00001} + \frac{1 + 0.00001}{0 + 0.00001}\right) \div 2 = 100,001 \tag{6.4}$$

Since *document three* has the higher score it is retrieved next. Intuitively, *document three* is pointed to by more terms that have been found to link to changed documents. This process continues until the crawl cutoff it reached and the crawl process then stops. As the crawl progresses, more statistical samples become available for each term, and so the change and

unchange statistics of each term become more accurate. At the end of the crawl, the complete change statistics for each term are known, and so this information can be incorporated into the ordering of the next crawl.

6.2 Dynamic Scheme Adaptation

In this section we describe a subtle modification that we make to our approach to ensure that it can adapt to situations where the effectiveness of term change statistics is limited. This is useful in situations where a site has a large disparity in its change frequency across different sections, such as in a large domain where different sections change with varying frequencies due to modification by different departments. For example, a university site that consists of several independent sections based on faculties or departments. Each departmental section has common terms such as "enrolments" or "events". In some departments these consistently link to frequently changing documents, while in others they consistently link to static documents. In this case, the likelihood of an anchor term linking to an altered document depends on the location of the anchor term within the context of the site.

Our Dynamic Scheme Adaptation (DSA) approach monitors the ongoing performance of both the default scheme and the dynamic anchor based scheme. It dynamically switches to the approach that has performed more effectively. This approach is also suitable for situations where we want to control the degree to which the default scheme affects the overall crawl strategy.

This modification allows our approach to perform almost as effectively as the default scheme when anchor term statistics are not effective, and also allows performance to improve when anchor statistics are effective.

6.3 Results

In this section we present the results of our experiments presented as the *crawl rate* versus the percentage of the collection retrieved. Crawl rate is the percentage of changed or new documents in the collection that have been retrieved so far, where change is defined using WORDS with an α change threshold of ten words. Plotting the crawl rate against the percentage of the collection retrieved demonstrates how well a scheme finds changed documents

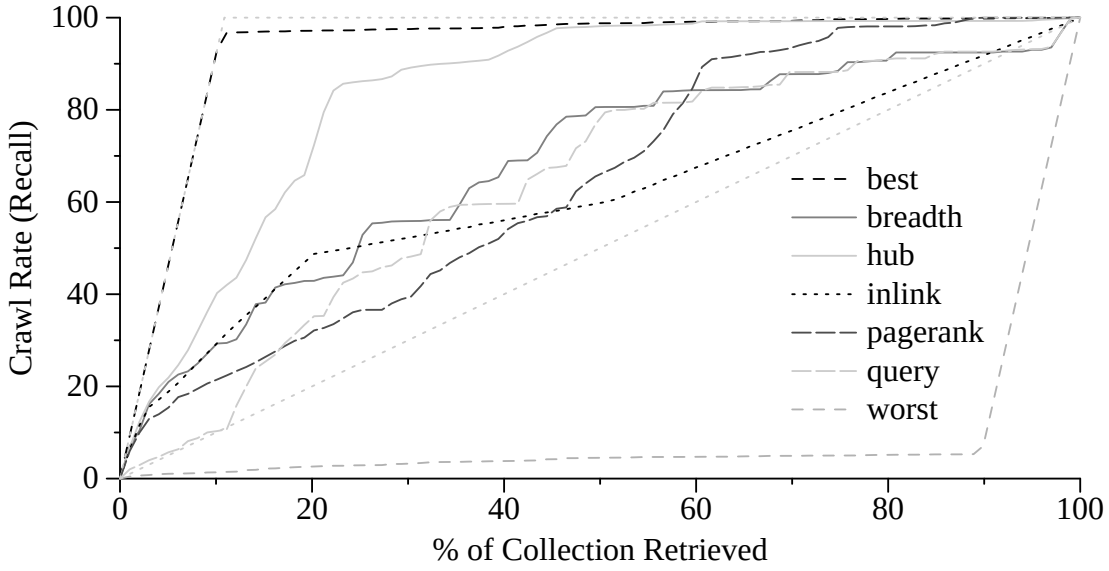


Figure 6.2: Crawl rate vs percentage of the collection crawled for the ABC collection on crawl two using default schemes.

during the crawl process. The graph is monotonically increasing, that is, the crawl rate starts at a value of zero and the value can only increase during the crawl.

In these graphs, it is important to note that the performance of a scheme during the beginning and particularly the middle of a crawl is most significant. That is, a good scheme will achieve a high crawl rate during the early stages of the crawl, allowing the crawl to stop earlier and avoid wasting resources retrieving static documents. While we only report results from crawl two, we have observed similar results during other crawl periods. Furthermore, each graph outlines the performance of the top five schemes — Breadth, Hub, Inlink, PageRank, and Query— as well as two control schemes — Best and Worst. The Best scheme orders a crawl so that it retrieves all significantly changed documents first, while the Worst scheme retrieves static documents first. Again, change is defined using the Words metric with an α change threshold of ten words.

Since some documents may change by less than ten words, yet contain links to new documents, the Best scheme does not necessarily capture all new documents before retrieving static documents. Similarly, the Worst scheme does not necessarily avoid all new documents before retrieving new and changed documents.

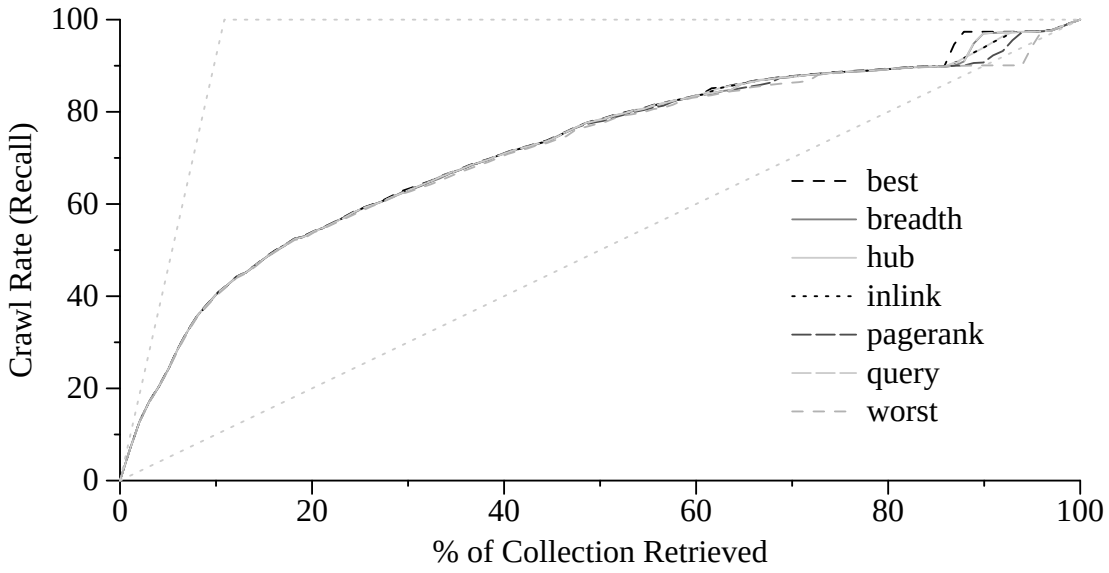


Figure 6.3: Crawl rate vs percentage of the collection crawled for the ABC collection on crawl two using term frequency.

We also plot an upper limit for performance and the performance in the average case, both represented by a grey dotted line.

In our graphs we plot four variations of the anchor based schemes: *phrase*, *term*, *term freq*, and *phrase freq*. With each scheme, any new documents that are retrieved are considered relevant, however, none of the schemes bias towards terms that retrieve new documents. Furthermore, schemes vary in the way that term statistics are calculated.

The *phrase* (S) scheme considers the entire anchor as a complete indivisible unit. Other anchors must match the entire anchor to share change statistics. The *term* (S) scheme breaks each anchor up into distinct terms and monitors their change statistics in the collection. The *term freq* (S_f) scheme is similar to the *term* scheme, in that it also breaks up each anchor into terms, however in addition, it keeps track of the frequency at which each term points to a particular document. The *phrase freq* (S_f) scheme similarly considers the entire anchor, while keeping track of the frequency with which a particular phrase points to a document. In addition, each scheme is based upon a default scheme (without any term statistics). That is, if all unvisited documents have the same score, the default ordering will take precedence for the scheme. Each of the four anchor based variation can thus be compared to this default scheme.

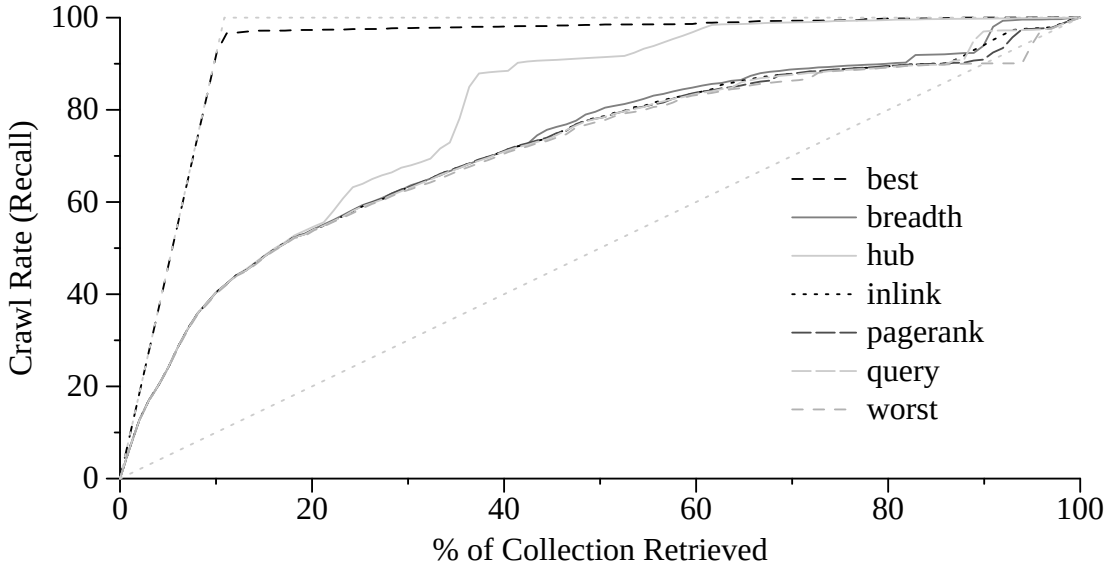


Figure 6.4: Crawl rate vs percentage of the collection crawled for the ABC collection on crawl two using term frequency with dynamic adaptation.

In Figure 6.2, we present the default performance of each scheme on the ABC collection without the influence of any anchor statistics. It is clear that the HUB scheme is most effective during the entire crawl and retrieves close to 100% of all changed or new documents after retrieving less than 50% of the collection. The performance of the remaining schemes fluctuates to varying degrees.

In Figure 6.3, we introduce the use of term frequency statistics — *term freq* (S_f) — and note that each scheme performs virtually identically. In general, all schemes improve in their performance during the first half of the crawl, except for the HUB and BEST schemes, which actually drop significantly. This is attributed to the domination of the term frequency approach over the default scheme and is caused by the large number of term statistics that are gathered during the crawl which help identify which document to retrieve next. It is only towards the end of the crawl, after about 87% of the collection is retrieved that term statistics can no longer distinguish which document will change. At this point, the default scheme is used to decide which document to retrieve, hence the variations noted.

While the term frequency approach produces an improvement when the default scheme is poor, we note a performance drop when the default scheme is effective. To counter this, we use the DSA approach, introduced in Section 6.2, and present the results in Figure 6.4.

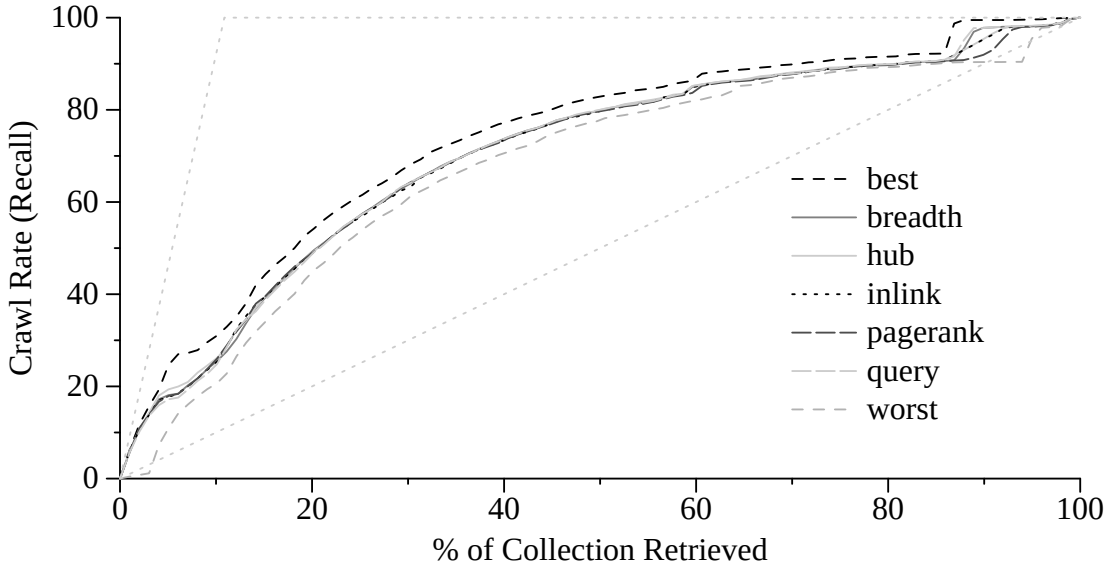


Figure 6.5: Crawl rate vs percentage of the collection crawled for the ABC collection on crawl two without term frequency.

We note that while the performance of the BEST scheme returns to optimal and the performance of the HUB scheme improves dramatically after retrieving 22% of the collection, the remaining scheme are not affected. After retrieving 22% of the collection using the term frequency statistics, the DSA approach detects that the performance has dropped to a lower level than that of the HUB scheme, and so switches schemes. At about 43% the DSA approach determines that the BREADTH scheme is performing more effectively than the term frequency approach and switches to that. Furthermore, since the BEST scheme always outperforms the term frequency scheme, the DSA it never switches from the default scheme. Finally, we note that the DSA approach does not alter the remaining schemes, since they do not perform as well as the term frequency approach for the majority of the crawl.

We note that while the PAGERANK scheme is actually more effective after 60% of the collection is retrieved, the DSA approach does not switch schemes because the PAGERANK scheme performed very poorly during the first half of the crawl.

We could easily make three improvement that we foresee would greatly boost the performance of the DSA approach. The first of these would be to weight the performance of schemes and the anchors approach, based on their recency. In this way, the recent performance of each scheme would have a greater weighting than its prior performance. This would

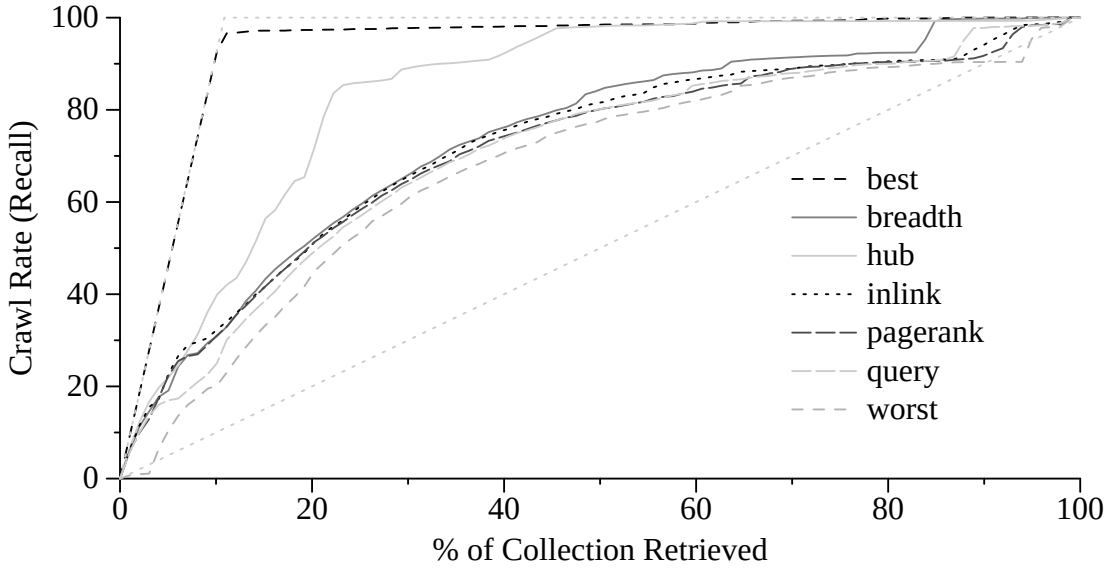


Figure 6.6: Crawl rate vs percentage of the collection crawled for the ABC collection on crawl two without term frequency using dynamic adaptation.

make the scheme selection more dynamic. In Figure 6.4, the DSA approach would switch to the HUB scheme after about 12% of the collection retrieved, and to the PAGERANK scheme after about 60% of the collection retrieved, since at this point the recent performance of the term frequency approach has dropped, while the performance of HUB and PAGERANK schemes continues to improve.

The second improvement would be to apply a measure of confidence on the reported effectiveness. This would be based on the recency of the last sample and the total number of samples. Hence, if a scheme hasn't been used recently, or frequently the DSA approach is more likely to switch to it. This would ensure that a scheme that performs poorly to begin with but improves dramatically later is not overlooked because there are no new samples highlighting this improvement.

Finally, we could use the DSA approach to select between all default schemes and the anchors approach. By allowing the crawl to switch between a large variety of crawl patterns, this hybrid approach would allow the crawler to adapt to a wide variety of domains more effectively.

In the next set of results, presented in Figures 6.5 and 6.6, we examine the use of anchor terms without the inclusion of frequency statistics — *term* (S).

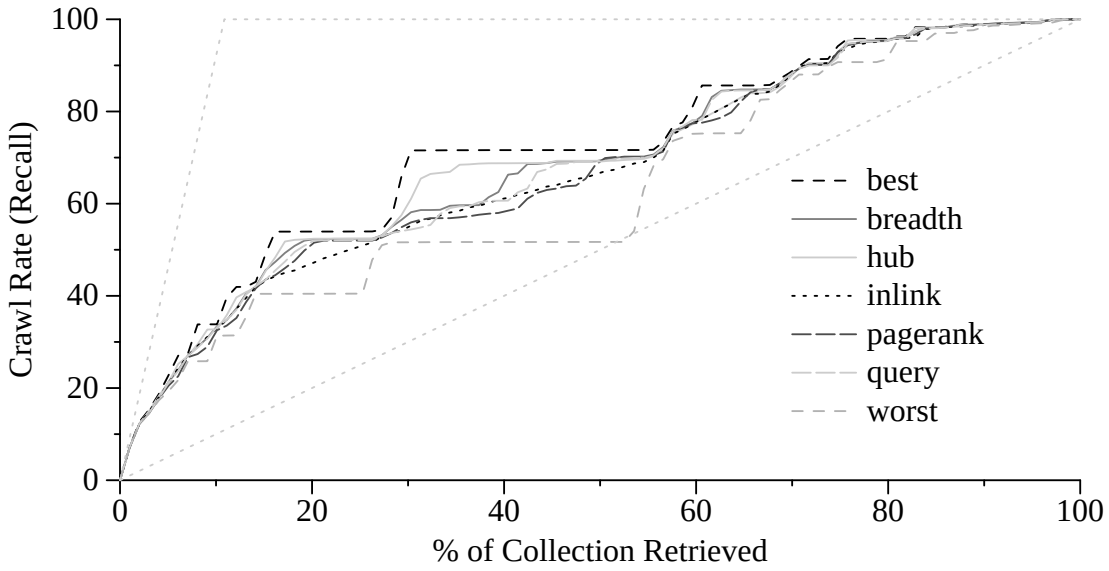


Figure 6.7: Crawl rate vs percentage of the collection crawled for the ABC collection on crawl two using phrase frequency.

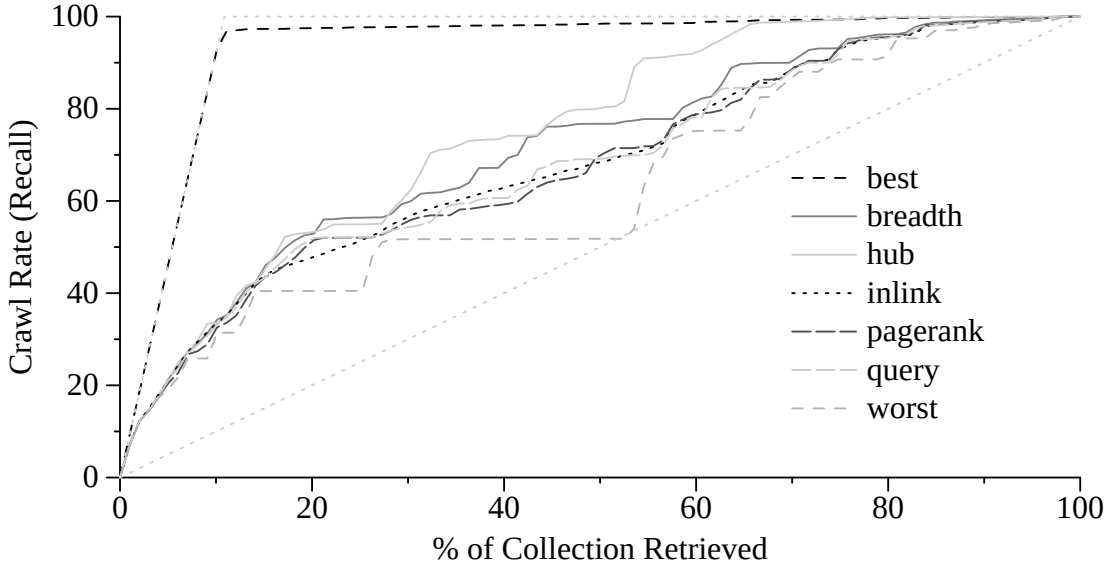


Figure 6.8: Crawl rate vs percentage of the collection crawled for the ABC collection on crawl two using phrase frequency with dynamic adaptation.

We note that there is a greater variation in the performance of the different default ordering schemes, highlighting that the anchor approach has less information with which to order the crawl. In this case, it relies upon the default scheme more heavily. This is clearly evident at about 16% of the collection crawled — at this point all schemes feature a plateau in the crawl rate, when compare to the results incorporating term frequencies in Figure 6.3.

It is also at this point in Figure 6.6, that the DSA approach detects that the HUB scheme is more effective than the anchor terms approach, and so switches schemes. Similarly, at 47% of the collection crawled, the DSA approach detects that the BREADTH scheme is more effective than the anchor terms approach, and switches schemes.

In Figures 6.7 and 6.8, we highlight the performance of the phrase frequency scheme — *phrase freq* (S_f) — both with and without the impact of the DSA approach. We can clearly see the impact of phrases on crawl rate, particularly the performance of the BEST and WORST control schemes. In the initial stages of the crawl, there are significant improvements in the crawl rate due to the discovery of new and changed documents. However, there are several large plateaus in crawl rate. The vertical improvements in crawl rate for the BEST scheme, especially during the middle stages of the crawl, are due to the crawling algorithm falling back on the default scheme in the absence of useful phrase statistics. With the introduction of the DSA approach in Figure 6.8, we note that the BREADTH, HUB, and BEST schemes have distinct alterations to their crawl rate. In the case of these schemes, the DSA approach detects that the effectiveness of the phrase approach has stalled, and shifts to the default scheme instead.

Finally, in Figure 6.9, we present the results of our schemes on the CSIRO collection. Since our previous results in Section 5.6 showed that all schemes essentially performed as effectively as the RANDOM approach, we compare the various anchor based approaches using the random default ordering.

We note that all schemes, including the default scheme perform optimally until about 3% of the collection is retrieved. Closer examination revealed that this was due to the discovery of new documents which, as we noted earlier, was caused by major restructuring of the CSIRO site during the crawl period. Importantly, all variations of the anchor based approach produce substantial improvements over the default RANDOM scheme.

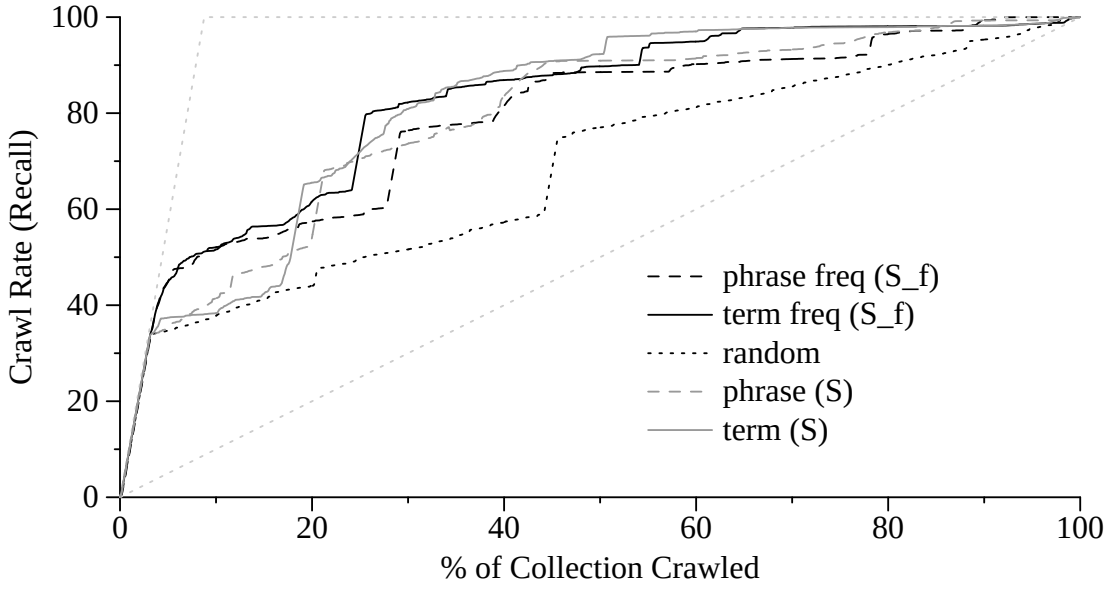


Figure 6.9: Crawl rate plotted against percentage of the collection crawled with a random default scheme using the CSIRO collection.

Furthermore, it is clear that the plots of the different anchor schemes share a similar shape to the default scheme, indicating that the anchor based approach is falling back onto the default scheme when the score for each page is the same. Hence, a compromise is made between improved document freshness and the default scheme objective. When combined with the PageRank default scheme, this ensures that the crawled documents are both fresh and important. This can be extended to the Query scheme to ensure retrieved documents are fresh and likely to be returned in response to user queries.

Overall, we note that the schemes that incorporate frequency typically tend to perform best — particularly during the beginning of the crawl — suggesting that a number of distinct terms occurring with a high frequency are a more significant indicator of the relevance of terms with respect to document change, than many different terms with fewer occurrences.

When it comes to identifying documents that change, having fewer terms that occur in high frequencies in the collection has a larger impact than lots of different terms occurring infrequently. Intuitively, this makes sense, PageRank relies on this same principle to determine document importance, while topic-driven crawling uses this idea to test for document

relevance to a theme. Of all the schemes, we find that the *term freq* (S_f) performs best during the beginning and middle of the crawl.

6.4 Analysis

Continuing with the analysis that we introduced in Section 5.7 (page 160), we now analyse the computation costs and bandwidth savings of our dynamic approach to crawling.

$$\begin{aligned} Cost &= (N \times N \times A \times N \times \frac{A}{C}) \\ Cost &= (N^3 \times \frac{A^2}{C}) \\ A &= \frac{\text{number of unique anchor links}}{N} \\ C &= \frac{\text{number of links}}{\text{number of unique anchor terms}} \end{aligned} \tag{6.5}$$

where N is the number of URLs, B is the average number of terms linking to each page, and M is the average number of documents linked to by each term.

Compared to the schemes we introduced in Chapter 5 (page 129) the adaptive schemes that we introduced in this chapter are much more computationally expensive, however the bandwidth savings (as measured by their effectiveness) are much more significant, as we have shown in the previous sections.

6.5 Summary

All crawl schemes that rely on a mathematical model require at least two crawls of the Web in order to fit a model of change, and preferably many more so that the model fit is accurate. The methods that we have proposed and tested in this chapter require only the previous crawl of the Web, but if required, can be maintained over many crawls to identify long term trends. Our results have demonstrated that our approach of using anchor term statistics is an effective means of predicting page change. Furthermore, our schemes are adaptive, and not reactive. That is, unlike traditional change-based crawling schemes which rely on past change statistics, our schemes use the statistics generated during the current crawl. This ensures that our schemes have the most accurate information available to make crawl decisions. They are not one crawl behind, and so, the statistics used reflect current page change dynamics,

not those of the past which may no longer be accurate. Our schemes use both positive and negative sampling to help them perform well whether combined with highly effective or poor schemes. Our schemes can be combined with different default schemes to produce a solution that meets multiple objectives. For instance our schemes can be combined with a PageRank or Query default scheme to ensure that retrieved pages are fresh, as well as important or popular.

In particular, we found that the *term freq* (S_f) scheme generally improved upon the default scheme and produced the most improvement of all the schemes evaluated, particularly during the first half of the crawl.

The use of terms and their frequency produced the most improvement. This was directly influenced by the fact that the number of distinct terms was high, while the number of distinct phrases was dramatically lower. Furthermore, the use of frequency statistics helped reinforce terms change dynamics. That is, a term that linking frequently to a particular document had a greater impact on the documents statistics than a term that linked only a few times, hence clearly identifying important change indicating terms.

Furthermore, the plots of each evaluated scheme generally maintained the overall shape of the default scheme, despite improvements in collection freshness, but particularly when anchor statistics were not effective. This indicated that the schemes were falling back on the default scheme, and so producing a collection that combined the properties of the default scheme as well as producing a collection that was generally fresher.

Finally, the use of our DSA approach allowed our schemes to adapt to situations where the default scheme began performing more effectively than the anchor based approach. We foresee greater improvements in performance of the DSA approach with the introduction of further modifications. These include the introduction of both a decay and a measure of confidence on crawl statistics, and the incorporation of several default schemes from which to select the most effective.

In the next chapter, we summarise our results and discuss future work.

Chapter 7

Conclusions and Future Work

In this thesis we have examined algorithms for web crawling and methods for evaluating these algorithms. In particular, we have studied the impact that different crawling strategies have on the freshness of a crawled collection, that is, the degree to which a crawled collection matches the content of live Web pages. While there have been extensive studies in the past that have investigated the impact that crawl ordering has on collection quality, much less work has been carried out on collection freshness. Much of the existing work dealing with collection freshness has focused on modelling changes in Web pages over time, typically according to a Poisson process. Past work has also largely ignored the impact that collection freshness has on the search engine user, and the impact that new documents have on search results, something that we explore throughout our work by examining the changes to users' query results.

Another important factor that has been largely ignored in past work, is the maintenance of collection freshness in cases where there is insufficient data to accurately fit a mathematical model for change, particularly during the initial recrawls of a page. While previous studies have examined the use of past change statistics for maintaining document freshness, this approach requires that all documents are recrawled at least once so that their changes can be modelled, and preferably recrawled for many crawls for the model to be accurate. Whilst this approach can be quite effective, it relies on past changes reflecting current change dynamics, since the model is always built on statistics that are at least one crawl old. Therefore, this approach does not work well for documents that change at irregular intervals. In contrast,

our work in Chapter 6 (page 167), determines change statistics during the crawl process and adapts the crawl accordingly. Hence, our approach would adapt to handle documents that change at irregular intervals.

In our work in Chapter 2 (page 25), we highlighted many of the issues related to crawling, not just those restricted to collection freshness, and demonstrated that despite the relatively short history of the Web and crawling, both have developed and grown substantially. However, we also highlighted the lack of work into improving collection freshness, particularly in the absence of a long history of change, the abundance of studies that have made assumptions about the change frequency of documents in an attempt to simplify the problem and produce solutions, and the lack of empirical studies into the incorporation of user feedback. We have addressed these issues throughout our work in Chapters 3–6.

Our work in Chapter 3 (page 95), highlighted the limitations of evaluating crawler performance directly on the Web, and the need for an evaluation framework.

In Chapter 4 (page 101) we examined various methods for measuring document change and defined the WORDS and SIZE metrics, two approaches for effectively measuring and predicting document change affecting search results.

We verified the skewed nature in which documents were returned as answers to user queries, supporting the results of others [Garcia et al., 2004]. In Figure 4.2 (page 109), we demonstrated that for our PROXY collection of 12,348 documents, 1,762 documents were returned in the top 100 results in response to 50 queries. The remaining 10,586 documents were never returned. Hence, maintaining the consistency of the top 1,762 documents would have a greater impact on the consistency of user search results than updating the 10,586 documents that were never returned in the top 100 results. This principle would form a major component of our schemes and evaluation metrics.

We used this principle to formalise a novel approach to measuring the impact that changes had on search engine users and investigated various change metrics using this approach. Our evaluation method measured the differences in ranked search results to determine how changes in indexed documents would impact the query results of search engine users.

We showed that the WORDS metric was effective as both a predictor and detector of changes affecting search results. It predicted 93% of variations in rank results while recrawling 22% of the collection. In contrast, the SHINGLING approach, which is used in many studies,

retrieved 3.5 times as many pages to achieve this same result. When it came to detecting changes, the WORDS metric was able to detect 97% of rank variations due to stale documents, while recrawling 22% of the collection. The SHINGLING approach once again required 3.5 times as many pages to match the effectiveness of the WORDS metric. Using the HTTP headers to detect change could capture 56% of all rank variations and would retrieve 9% of the collection, matching the performance of the SHINGLING approach for the same number of updated documents over a one crawl interval. The HEADERS approach, however, performed quite badly at predicting change, detecting 9% of variations while retrieving 14% of the collection. In contrast, the WORDS approach predicted over 52% of rank variation for the same number of documents updated.

Through our work in Chapter 4 (page 101), we identified WORDS as an efficient and effective scheme for finding significant changes in documents. We adopted the WORDS metric with an α change threshold of 10 words throughout our work in later chapters. We also highlighted the data collection which we generated and used throughout our work.

In Chapter 5 (page 129) we highlighted the fact that many stateless schemes have been used in the past to determine their effect on collection "quality" without considering their impact on collection freshness. We set out to determine how these schemes impacted on collection freshness by recrawling a restricted number of documents from two collections on a weekly basis. Our evaluation method also ensured that no past change statistics were available to any of the schemes. We introduced the concept of popularity, a new evaluation constraint that determined how frequently documents were returned as part of a result set in response to user queries.

Few studies in the past had taken new documents into consideration when evaluating crawl schemes. We incorporated new documents by evaluating their retrieval as a success, introduced a new ordering scheme QUERY which sorted documents by the frequency that they were returned in response to user queries, and introduced HUB, a simplified adaptation of the HITS algorithm.

We found that while certain schemes were shown to perform well at retrieving important documents in other studies, they did not perform well at retrieving changed documents. Our results demonstrated that the most effective scheme was highly dependent on the collection being crawled. Specifically, for the ABC collection we found that the HUB scheme was most

effective and its performance was statistically significantly better than a random crawl. On the CSIRO collection, we found that the URLLENGTH scheme was most effective in retrieving changed and new documents. However, when we applied the additional constraint of popularity, we found that no scheme performed statistically significantly better than the random crawl. We suspected that this was because there was no correlation between popularity and change.

We were able to confirm this during our experiments with the QUERY scheme — our approach to updating popular documents based on the frequency that they were returned in response to user queries — we found that it performed poorly, producing a result that was essentially random from a freshness perspective. There was no correlation between popular documents and changed documents. This demonstrated that freshness and popularity were competing goals, in much the same way as quality and freshness were shown to be.

We also investigated the stability of schemes over each crawl interval, demonstrating that the performance of different schemes was stable over different intervals. That is, a scheme that was found to perform well during earlier crawls of a site would perform well during later crawls for the period of at least eight weeks.

When we evaluated the impact that various schemes had on user search results, we found that while the HUB scheme still performed well for the ABC collection, the QUERY scheme outperformed all other schemes. This highlighted the need to incorporate query statistics in the crawl process. Importantly, the skewed nature of query results emphasised the significant impact that changes to relatively few documents could have on query results.

Our results in Chapter 5, led to our work in Chapter 6 (page 167), where we aimed to produce a dynamic approach that could be combined with any stateless approach to produce a general scheme that combined the properties of both schemes. The approach that we developed used anchor term statistics to determine the likelihood that terms linked to changed and static documents. This approach was combined with a stateless default scheme. Whenever terms statistics could not identify which document in the frontier was most likely to change, the default scheme was used to order the frontier instead. Furthermore, we ran our experiments were run over multiple intervals, with a decay on the weight of term statistics from past crawls to reduce their impact on the crawl order. We found that the performance of schemes remained fairly consistent.

This approach had several distinct advantages over other approaches. Since anchor terms were used rather than document terms, a much smaller set of terms needed to be maintained. This resulted in reduced processing and maintenance costs. Our approach did not require two complete crawls to order the crawl effectively and statistics from the current crawl were used rather than from past crawls, thus reflecting the current dynamics of pages. The approach did not rely on change models like many past schemes. The improvement in collection freshness was not restricted to the performance of the default scheme, since it used both positive and negative results to improve freshness. It could be easily combined with any stateless scheme to achieve a variety of objectives independent of collection freshness, while still improving collection freshness. This approach produced improvements in collection freshness despite changes in the collection dynamics and topology, a factor which had adversely affected the stateless schemes we investigated in Chapter 5 (page 129).

Overall, we found that our adaptive scheme *term freq* (S_f) generally improved upon the default scheme. With the introduction of our Dynamic Scheme Adaption (DSA) approach we saw a marked improvement in collection freshness in the cases where the default scheme outperformed the anchor based approach. In the situation where no anchor statistics were available, the approach would fall back to the default scheme ensuring that the collection maintained the properties of the default scheme while improving collection freshness.

In our results on the CSIRO collection, we noted that while there was an improvement in collection freshness, the overall shape of the plot of the default scheme could be observed in the plot of the dynamic scheme. This indicated that the default scheme was having a significant impact on the documents that were being retrieved, despite the fact that there was an improvement in the freshness. Hence, the crawled collection maintained the properties of the default scheme while also being fresher. That is, if the default scheme was suited towards finding important pages, the retrieved collection would contain pages that were both important and fresh.

While our dynamic, anchor text based approach produced a marked improvement in collection freshness, there was an additional computation cost. Most of the stateless crawl ordering schemes that we evaluated in Chapter 5 (page 129) had a cost generally ranging from $O(N)$ for simple schemes such as Breadth and Depth to $O(N \log N)$ for most other schemes, where N is the number of documents that are ordered. Of all the stateless scheme

that we evaluated, QUERY had the highest computation with a cost of $O(N^2)$. In contrast our dynamic approach had a cost of $O(N^3 \times \frac{A^2}{C})$, where A is the average number of unique anchor terms linking to each document and C is the average number of documents linked to by each unique term.

While this is substantially higher than most of the stateless schemes we evaluated, it is important to note that other advanced crawl ordering approaches, such as those used by topic driven crawlers would also have a similar cost, and that their associated cost has not discouraged their use. In our case, the additional computation costs are likely to be outweighed by the bandwidth savings.

From our work, we can make the following observations:

- Crawling is difficult: Careful rules need to be defined to ensure that resources are not wasted, collections are properly maintained, and users are satisfied with their search experience.

- Crawler evaluation metrics should consider the skewed nature of search results and the impact that schemes have on search results: Whether evaluating change metrics, the performance of a crawl ordering scheme or any other aspect of crawling, the evaluation metric should in some way take into account the fact that query results are skewed and the impact that the scheme has on query results.

- Using past queries to predict change is ineffective: The set of documents that are frequently returned in response to user queries do not necessarily correlate with documents that change frequently, and so re-crawling them is a waste of resources.

- Using anchor terms to predict change is effective: There is a clear correlation between anchor terms and document dynamics, in much the same way that there is a correlation between anchor terms and document topics, a property used extensively in topic driven crawling.

Throughout our research we have highlighted the limitations of using HTTP headers to predict documents change, and the effectiveness of our WORDS scheme for both predicting and detecting change, particularly against the common SHINGLING approach. We have proposed a novel framework for evaluating the effectiveness of crawling, and demonstrated the

performance of the common crawling algorithms with regard to collection freshness, something that has been lacking in past literature. We have developed a novel approach for predicting document change based on the use of anchor terms. Furthermore, we have incorporated a dynamic method for selecting crawl algorithms using past observations of their performance as a selection criteria. Our approaches have focused on schemes that operate without the need for a past history of change. This makes them ideal for cases where this information is not available, particularly when documents are recrawled for the first time or new documents are found. Finally, our experiments have not relied upon assumptions or models of change on the Web, and instead we have used real world data to evaluate our schemes.

While we have endeavoured to minimise the impact of limitations, there are several key restraints within our work. Specifically, our framework was limited to eight and eleven weeks of crawls from the ABC and CSIRO domains respectively. Furthermore, our crawls were conducted on a weekly basis, and so we cannot draw any conclusions for changes that occur more frequently than once per week, or less frequently than once every eight to eleven weeks. Our crawls were limited to pages within these two domains, and so we have no anchor, term or link statistics from pages outside of these domains. External link information is likely to be more varied and objective, giving us a clearer understanding of the impact of anchor terms and their utility for classifying document change. Hence, a larger more diverse crawl would help us to evaluate our schemes more thoroughly. We discuss this and many other future avenues of investigation within the scope of our research in the next section.

7.1 Future Work

While we have examined many different facets of maintaining the freshness of crawled collections, this is by no means a solved problem, and there are many improvements and unexplored avenues that can be investigated.

Specifically, in our work in Chapter 5 (page 129), an avenue for further study would be the impact that document duplication has on performance, for instance, investigating any relationship between document change frequency and near-duplicate documents. That is, to examine the issue of change amongst loosely coupled mirrors.

While our popularity evaluation metric relies on binary scores — a document is either popular or unpopular — an alternative approach that could be adopted is to incorporate a continuous scale for popularity, so that resources are weighted according to their popularity. This approach would ensure that popular documents are weighted higher, and so their impact when retrieved or missed would be greater than for unpopular documents. This approach, however, would require the normalisation of scores so that the effectiveness would not exceed 100%.

Another area of investigation is the dynamic selection of a crawl cutoff value. This would require a method for predicting when to stop the crawl instead of using a hard cutoff and could be based on the ratio of changed, new, and unchanged pages. This would be very similar to our DSA approach, monitoring ongoing performance and deciding at which point to terminate the crawl.

Our work in Chapter 6 (page 167) could be extended further by incorporating IR measures such as *tf.idf*. This would weight the ability of terms to discriminate between documents and could be used to reduce computation costs. The impact of weighting would probably be marginalised since the change versus unchange ratio would tend to cancel out for terms that are randomly distributed amongst changed and static documents, but the computational costs would be significant.

Other methods for reducing the computation costs that can be investigated include reducing the frequency of term statistic updates. In our current implementation, term statistics are updated after each page is retrieved. This could easily be modified to update on every fifth page, for example. Different approaches would obviously result with varying degrees of impact on efficiency and effectiveness. With the inclusion of further optimisation techniques, our work could be extended to larger, more diverse collections, and our dynamic approach could be adapted to work with terms from these general collections.

Reducing the frequency of term statistics updates and reducing computational costs would allow us to use other means of evaluation. Specifically, we could incorporate our novel approach to evaluating change scheme performance, introduced in Chapter 4. While this approach was suitable for evaluating performance at reasonably large intervals, it was not suitable for use in our work in Chapter 6, since we evaluated the performance each time

we retrieved a document. In the case of our experiments in Chapter 6, we would have to re-evaluate about 18,000 queries, a total of 500,000 times for each scheme.

While our work currently incorporates a dynamic approach for freshness, this could be extended to include multiple factors such as freshness, popularity, importance, and term frequency in the collection. These factors would then be weighted with tuning parameters to allow for a specific focus. Again, this can easily be implemented in our DSA approach by combining several default schemes that possess these properties.

As we highlighted earlier in Chapter 6, we could easily improve the performance of our DSA approach through several simple modifications; Implementing a decay on the performance of the schemes and the anchors approach, applying a measure of confidence on the reported effectiveness and combining several default schemes. These measure would allow the DSA approach to be more dynamic in its selection process and allow it to adapt to different domains more effectively.

Our current implementation of the anchors based approach considers new and changed documents equally. This could be altered by dividing the term scoring system into two distinct components, new and changed. The crawl could then be biased towards retrieving documents that have changed or towards documents that link to new documents.

Our work could also be extended from anchor terms to incorporate the terms contained within documents in order to determine the content of changing documents, although this would be much more computationally expensive, and require more IR optimisations to reduce costs. Anchor terms could also be extended to include the terms surrounding them, in the same way that topic driven crawlers operate. In addition, terms that form the URL could also be incorporated. These could then be used as part of terms associated with change for documents. However, these terms would need to be collated and weighted separately since there would be fewer URL terms than anchor terms for each document. Furthermore, combining these terms with the anchor terms could "dilute" their ability to discrimination between documents that change and those that are static. For instance, in the URL's

(CHANGED) http://www.abc.net.au/**news**/**latest**/s12346.htm
(STATIC) http://www.abc.net.au/**news**/**old**/s12345.htm

the terms “news”, “latest”, and “old” could be extracted. Generally, the term “news” would be expected to be associated with many dynamic pages. In this case, however, the term “news” would not be a good change predicting term, since both URLs contain the term, but only one document has changed. Instead, in this particular case, the terms “latest” and “old” are impacting upon the performance of the term “news” as a change prediction term.

Appendix A

Crawler Design and Implementation

In this section, we discuss the problems and challenges that have come up during our development of a distributed crawler for collecting web documents for the Zettair search engine. We also present the solutions that we have implemented in response to these various problems.

A.1 Introduction

A web crawler operates by retrieving a resource from a web site, extracting the HTML links out of the resource, queuing each URL into its list of URLs to visit, storing a copy of the resource in a central repository, and then visiting the next URL in the list. The crawler stops retrieving resources once the list of URLs to visit is empty, or the central repository is full. Of course, if a crawler used a simple algorithm like the one just described, it would quickly begin retrieving multiple copies of the same resources. This would waste a large amount of bandwidth and flood web servers with unnecessary requests. The crawler therefore needs to keep track of URLs it has previously extracted from documents to avoid retrieving them again.

To complicate matters further, the URL to a particular resource can be represented in many different formats. The following URLs, for example, all return the same resource:

`http://www.cs.rmit.edu.au/~hali`

`http://www.cs.rmit.edu.au:80/~hali`

`http://www.cs.rmit.edu.au/%7Ehali`

`http://WWW.CS.RMIT.EDU.AU/~hali`

To determine whether a URL has been retrieved before, the crawler must convert each URL to a standardised format, before comparing them against the list of extracted URLs. Further to this, standardising URLs and ensuring that each unique URL is retrieved only once does not necessarily guarantee that resources won't be wasted. Web crawlers can get caught in what are commonly known as 'crawler traps' [Heydon and Najork, 1999]. Crawler traps are a set of resources that cause the web crawler to traverse them indefinitely. They are typically dynamically generated and of limited use to search engine users. They can be created maliciously in an attempt to obtain higher rankings in search results or to trap misbehaving crawlers. They can also be created unintentionally through dynamically generated resources, for example, an online diary with a document for each month, each with a link to the next and previous month. Obviously, if a crawler attempted to retrieve every entry in the diary it would never stop crawling and so, the crawler needs to either detect when it is in a crawler trap or avoid them altogether.

While crawlers are essential to the utility of the internet, they can also be a nuisance to web administrators. A badly behaving crawler may flood a server with a large number of requests in a short period of time, producing what is known as a Denial Of Service (DOS) attack, whereby the web server cannot respond to any other web user requests because it is inundated with frequent requests from a web crawler. Further, web crawlers may retrieve sensitive material that web publishers may not want to be indexed, or may even intentionally retrieve e-mail addresses for the purposes of unsolicited e-mail advertising known as "spam". In response to some of these issues, guidelines have been produced that promote good crawling practices and etiquette that should be observed by crawlers.

Previous work on crawlers have mainly concentrated on scalability [Heydon and Najork, 1999; Shkapenyuk and Suel, 2002], crawl ordering [Cho et al., 1998], parallelization [Cho and García-Molina, 2002], index freshness [Edwards et al., 2001; Ali and Williams, 2003] and topic-driven crawling [Chakrabarti et al., 1999; Diligenti et al., 2000; Chau and Chen, 2003].

In this section we present a variety of crawler related problems, and the solutions we implemented to overcome them in order to develop an efficient and effective crawler.

That is, a crawler that does not needlessly retrieve web resources, yet still manages to retrieve any resources that will be useful to the end user.

```
<HTML>
 <HEAD>
  <BASE HREF="http://abc.net.au/new/">
  <TITLE>Sample document</TITLE>
 </HEAD>
 <BODY>
  <P>Here is a <A HREF="http://abc.net.au/"> link </A>
    to the main page</P>
  <P>Both this <A HREF="index.html"> link </A> and this
    <A HREF="../index.html"> link </A> will use the Base URL
    to form a fully qualified URL</P>
  <A NAME=end></A>
 </BODY>
</HTML>
```

Figure A.1: An example of HTML. (sample.html)

A.2 Standardising URLs

One of the biggest challenges in developing a crawler is extracting the URLs from HTML documents and ensuring that they are standardised so that multiple copies of the same resource are not retrieved. In Figure A.1, we present a sample of HTML code from the URL

```
http://abc.net.au/test/sample.html
```

The HTML links in the resource are represented as anchor tags in the format

```
<a href="http://abc.net.au/">link</a>
```

The URL in the anchor tag above is represented in its fully qualified form, while the URLs in the following anchor tags are represented in their relative form.

```
<a href="index.html">link</a>
<a href="../index.html">link</a>
```

Normally these would be combined with the resource URL to form the following URLs respectively

```
http://abc.net.au/test/index.html
http://abc.net.au/test/../index.html
```

however in this case the resource contains the base URL

```
http://abc.net.au/news/
```

and so, once converted to their fully qualified form, produce the following URLs instead.

```
http://abc.net.au/news/index.html
http://abc.net.au/news/../index.html
```

Although the second URL will resolve correctly, it is important that it is reduced further so that the URL can be uniquely identified. Once fully resolved it becomes the URL

```
http://abc.net.au/index.html
```

Each relative URL in the HTML resource must first be converted to its fully qualified form as outlined above, before it can be uniquely identified.

URLs can also contain hexadecimal representations of characters. This is done by placing the "%" character before the hexadecimal value of the ascii character. For instance, the "∼" character can be represented by the sequence "%7e".

This notation allows web developers to embed characters that normally cannot be represented in a URL. Any character can be represented in this format, and so it is important that they are converted in a standardised way. This is made more complicated by the fact that certain characters, like the space character, can only be represented in hexadecimal notation, while the rules for other characters depend on where in the URL they occur. The first occurrence of the "?" character for instance, must always be converted to the character format, while all subsequent occurrences must be converted to the hexadecimal representation "%3F".

This is to ensure that all characters after the first occurrence of the "?" character are parsed as variables including subsequent occurrences of the "?" character. In addition, because this format is not case sensitive, both "%7e" and "%7E" will produce the "∼" character. Hence, these rules must be clearly defined by the crawler so that there is no ambiguity in the URL format.

HTTP URLs typically consist of several key components. The domain or host, path, resource, and variables.

$$\{Domain\}\{Path\}\{Resource\}\{Variables\}$$

```
http://abc.net/files/file.php?p=2&q=books
```

The domain identifies the web server where the resource resides. The path identifies the absolute directory path to the resource on the server. The resource identifies the file that is being requested. Finally, variables are user parameters that are passed to executable resources. Each variable identifier and value pair is separated by an ampersand "&".

An important rule with URLs is that while the resource and path components of a URL are case sensitive, the domain is not. The following two URLs, for example, are the same.

```
http://abc.net.au/index.htm
http://ABC.NET.AU/index.htm
```

While, the following two URLs are different.

```
http://abc.net.au/test.html
http://abc.net.au/TEST.HTML
```

This in turn means that the domain of a URL needs to be case folded in a standardised way, for instance, by converting all characters to lowercase, while the resource and path must remain unchanged.

Occasionally, URLs also include a port address, typically because the port is non standard. Any URL that does not specify a port address will use the default port 80, however the default port address can be explicitly included in the URL. The following URLs for instance, will return the same resource.

```
http://www.cs.rmit.edu.au:80/
```

```
http://www.cs.rmit.edu.au/
```

And so, if the default port (80) is included in the URL, it must be handled in a consistent manner, by either removing it in all cases, or including it in URLs when no port is specified.

Web publishers can also produce in-document links, that is, links to specific sections within a HTML document, by using the "`#`" character in a URL and placing a corresponding anchor tag in the document. For example, the URL

```
http://abc.net.au/test/sample.html#end
```

links to the "`end`" tag in the sample.html document from Figure A.1. However, the returned resource does not change with the inclusion of the "`#`" character, therefore, to prevent redundancy, any suffix beginning with the "`#`" character must be removed. And so, once altered, the previous example becomes.

```
http://abc.net.au/test/sample.html
```

Standardising URLs in the ways outlined in this section prevents the crawler from retrieving the same resource multiple times due to different representations of the same URL, however there are other factors that affect the crawling process.

In the next section we highlight the HTTP interaction process and how this affects the crawling process.

A.3 HTTP Interaction Process

The HTTP standard allows clients to request resources from different servers in a standardised way, and so, all HTTP transactions must follow the HTTP interaction process, outlined in Figure A.2.

1. The process begins with a connection request from the client (web browser or crawler) to the web server, typically on the default port 80.

2. The server then responds with either an acknowledgement indicating that the connection was accepted, or a rejection indicating that the request was denied and the connection closed.

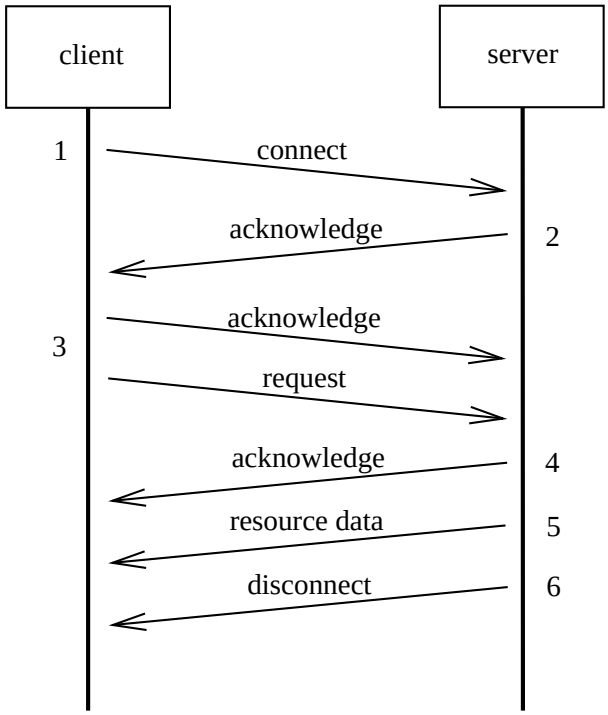


Figure A.2: The HTTP interaction process. The process begins with a connection request from the client to the server (1). The server responds with either an acknowledgement accepting the connection, or a rejection denying the connection (2). The client responds with its own acknowledgement and then sends a HTTP header requesting a resource from the web server (3). The server follows up with an acknowledgment to the client and then responds with its own HTTP header (4). The requested resource is then sent (5). Finally, the server closes the connection (6).

```
GET /test/sample.html HTTP/1.0
Host:  abc.net.au
Accept:  text/html
User-Agent:  Wget/1.9
Connection:  close
```

Figure A.3: An example HTTP request.

```
HTTP/1.1 200 OK
Date:  Mon, 25 Oct 2004 10:58:53 GMT
Server:  Apache
Set-Cookie:  ABCGuestID=131.170.24.156.1098701933415944; path=/;
             expires=Mon, 01-Nov-04 10:58:53 GMT
Accept-Ranges:  bytes
Cache-Control:  max-age=-453594
Expires:  Wed, 20 Oct 2004 04:58:59 GMT
Connection:  close
Content-Type:  text/html
```

Figure A.4: An example HTTP response header.

3. When the client receives the acknowledgement it responds with its own acknowledgement indicating that the server's acknowledgement was received. It then sends a request for a resource from the web server in the form of a HTTP header, as shown in Figure A.3.

In the example, shown in Figure A.3, the request is for the resource "`/test/sample.html`" (along with the HTTP response headers) using the HTTP 1.0 protocol. Alternatively, the client could request only the HTTP response headers by using the HEAD command, instead of the GET command.

The HTTP request must also specify which host it is requesting the resource from, since the web server could potentially host many different virtual domains, all using the same IP address. In this particular case the host is identified as "`abc.net.au`". The third line in the

request, indicates the file format that the web client supports, while the fourth line in the request identifies the web client to the server. The final line indicates whether the server should keep the HTTP connection open after the current request is completed, so that the client can make further requests. In this particular example, the connection will be closed after the resource is transmitted to the client. It is important that each line is separated by a carriage return followed by a line feed "`\r\n`", with an empty carriage return line feed to end the request, as outlined in the HTTP specification [Fielding et al., 1999a].

4. After receiving the clients request, the server follows up with an acknowledgment to the client and then responds to the clients HTTP request with its own set of HTTP headers, as shown in Figure A.4.

5. This is followed by an empty line and then the requested resources, in this case the HTML document "`sample.html`" shown in Figure A.1.

6. The server then closes the connection.

The first line in the HTTP response header, shown in Figure A.4, indicates the version of the HTTP protocol being used for the response as well as a resource status code. The code is a three digit number that indicates whether the requested resource was successfully returned. A number in the 200–299 range indicates that the requested resource was successfully returned to the client and follows on after the headers. A code in the 300–399 range indicates that the requested resource has moved to another location. This is typically accompanied with the "`LOCATION`" header field indicating the new location. A code in the 400–499 range indicates that there is a client side error, for instance, a 404 code indicates that the requested resource does not exist on the web server. A code in the 500–599 range indicates there is a server side error.

The next field, "`DATE`", indicates the web servers local date and time. The "`SERVER`" field identifies the web server type and version (typically). The "`SET-COOKIE`" field is used by the server for state maintenance. The "`ACCEPT-RANGES`" field indicates whether the server can handle request for only part of a resource. Both the "`CACHE-CONTROL`" and "`EXPIRES`" fields are used for maintaining cache consistency and coherence. The next field, "`CONNECTION`", indicates whether the connection will be terminated after the current communication ends.

```
<HTML>
 <HEAD>
  <META HTTP-EQUIV=refresh CONTENT=2;url=http://abc.net.au/>
 </HEAD>
</HTML>
```

Figure A.5: An example of a redirection using META tags. The user will be redirected to the URL "`http://abc.net.au/`*" after a period of two seconds.*

Finally the "`CONTENT-TYPE`" field indicates the file format of the resource that will follow on after the header. Search engines typically index only HTML documents, and so crawlers need to be able to determine resource formats before they are retrieved. A convenient approach is to use this HTTP response header field. By using a HEAD request rather than a GET request, the crawler can retrieve both the status code and content-type to determine the availability and type of resources, without needlessly downloading large resources that search engines cannot index. At the same time it can use the headers to determine if there is a page redirection, which we highlight next.

A.4 Page Redirection

Page redirection occurs when a resource redirects the user to another page automatically through the use of HTTP response headers or HTML META tags. As outlined earlier, the web server can return a HTTP response header with a code in the 300–399 range indicating that the resource has moved. This is accompanied by the "`LOCATION`" field indicating the new URL where the resource can be found. An alternative method of redirection is through the use of HTML META tags. The example in Figure A.5 includes a META tag in the HTML that instructs the web client to redirect to the URL "`http://abc.net.au/`" after two seconds. In both cases, the web client must create a connection to the new domain identified in the URL and then request the resource in the URL.

In order to avoid continuously resolving these redirections, any URL that causes a redirection must also be added to the seen URL list.

Occasionally, crawlers come across recursive URL redirections, that is, resources that redirect to themselves. This problem is solved by comparing redirected resources against the seen URL list.

In addition to redirections, web servers also define a default resource that is returned if none are specified in the URL. The following URLs could potentially return the same resource, since in many cases, if no resource is specified, the default resource "`index.html`" is returned instead.

```
http://abc.net.au/index.html
http://abc.net.au/
```

There are, however, many different standards for default resource naming. On most Apache installations the default resource is "`index.html`", while on most IIS installations the default is "`home.html`", "`default.html`", or "`index.html`". The file extensions are typically "`.htm`", "`.html`", or "`.php`", however the default resource can be customised to anything the web administrator chooses. One simple approach to handle this is to include rules for the most common default resources (index.xxx, home.xxx, and default.xxx), while a more robust approach is to request the resource and then determine the resource that is returned via the HTTP headers.

Another similar problem is the addition of trailing slashes in URLs. For example, the following two URLs are identical.

```
http://abc.net.au/
http://abc.net.au
```

Although many servers resolve this through redirection, this is not always strictly the case, and so it is important that the crawler keep track of this. One simple approach is to remove trailing slashes from URLs before they are inserted into, or, compared against, the seen URL list.

During a multitude of test crawls, we also came across several peculiarities. We found that URLs could contain consecutive slashes, while the domain could contain a trailing dot. In both cases the URLs would resolve correctly. The following three URLs, for instance, all resolve to the same resource.

```
http://abc.net.au///index.html

http://abc.net.au./index.html

http://abc.net.au/index.html
```

These issues are easily solved by converting multiple consecutive slashes into a single slash (except for the double slashes preceding the domain name), and by removing any trailing dots in the domain.

On several occasions, domains returned different response headers when a HEAD request was made instead of a GET request. A HEAD request would return headers with a "`404 Not Found`" error, while a GET request would return a "`200 OK`" header. Our approach to dealing with this was to first issue a HEAD request. If this returned a "`200 OK`" or a non-html resource type, then the header was considered valid and processed accordingly. Otherwise, a GET request was issued instead to confirm the validity of the headers.

Another interesting observation was that many domains returned customised "`404 Not Found`" error documents when requested resources could not be located, however the headers contained a "`200 OK`" response code. This is also known as a soft 404, and a heuristic for their identification has been developed by Bar-Yossef et al. [2004]. In the next section we highlight the problem of crawler traps, and how they can be avoided.

A.5 Crawler Traps

Crawler traps are another potential hazard for web crawlers. Crawler traps are resources that can potentially be traversed indefinitely and are typically dynamically generated [Heydon and Najork, 1999]. They can be created maliciously to make a site appear to be large, popular and highly linked, in an attempt to increase its presence in search results. They can also be created unintentionally by web site administrators inadvertently creating a recursive symbolic link, for instance, by creating a directory with a symbolic link to the parent directory. Another common crawler trap is created by dynamically generated content that does not end. For instance, entries in an online diary that goes on indefinitely into the future, or a site that calculates the value of π and allows the user to move along the sequence.

In our implementation, we prevent crawler traps by restricting the number of documents that the crawler retrieves from a particular dynamically generated resource. For instance, the following URL

```
http://go.com/diary.php?d=10&m=02&y=2004
```

could be reduced to the dynamic resource

```
http://go.com/diary.php
```

by removing the suffix beginning with the "?" character. The crawler is then restricted to retrieving a limited number of variations of the same dynamic resource. Another alternative would be to retrieve only the reduced dynamic resource without any parameters. Finally, the crawler could restrict the dynamic resources that are retrieved to only those that are linked to by static resources, by not traversing any dynamic URLs that are in a dynamic resource themselves. Since static URLs cannot dynamically generate links or resources, the crawler cannot crawl them indefinitely.

Another related problem is URL rewriting. URL rewriting is a server side function that converts the URL of dynamic resources into a static format at runtime. For instance, URL rewriting would convert the URL

```
http://go.com/diary.php?d=10&m=02&y=2004
```

into the URL

```
http://go.com/diary/10/02/2004
```

The parameter values are passed as a directory structure, while their order determines the variables being set. URL rewriting makes distinguishing dynamic resources from static resources difficult, particularly for crawler trap avoidance.

Next we reintroduce mirroring and explain the impact that they have on crawling.

A.6 Mirrors

As we described in Section 2.13 (page 85), mirrors are a set of resources that have been replicated on different domains, typically because they are on a different physical machine to

help distribute server load and improve performance. It is important that crawlers detect and avoid crawling mirrors as they do not provide any additional benefit to search engine users, but require a large amount of resources and bandwidth to crawl. One method of detecting mirrors is to detect similarity in URL resource paths across different domains.

For example, the following URLs could potentially be mirrors since there are similarities in their directory structure.

```
http://go.com/books/c/comp/list.html
http://test.com/books/c/comp/list.html
http://a.com/cp/books/c/comp/list.html
```

Another method is to determine content similarity. The easiest and fastest way to detect identical documents is to create a document fingerprint using a scheme such as an MD5 hash [Rivest, 1992], and to compare the fingerprints. However, since changes made to the original site tend to trickle slowly to mirrors, there is usually some difference in content across mirrors. Since even the slightest difference in content will generate a completely different document fingerprint, the similarity of mirrored documents would not be detected. A better approach is to use a change metric, such as shingling [Broder et al., 1997] or our own metric WORDS, introduced in Chapter 4 (page 101), and allow for a small amount of differences. This allows resources that are identical or nearly identical to be detected across different mirrors.

While there are many different approaches to detecting mirrors, no one single method alone can detect all mirrors. However, by combining URL resource similarity with content similarity across an entire site, a crawler can help reduce the likelihood of needlessly crawling mirrors [Bharat and Broder, 1999]. In our work we do not target mirrors, and so our implementation does not take mirroring into account.

In the next section we introduce the difficulties associated with maintaining crawler etiquette.

A.7 Crawler Etiquette

Several studies in the past have highlighted the impact of badly behaving crawlers, and have demonstrated approaches toward good crawler etiquette [Koster, 1993; 1995; Dill et al., 2002].

```
User-agent:  Mediapartners-Google*
Disallow:

User-agent:  *
Disallow:  /forum
Disallow:  /pm
Disallow:  /search
Disallow:  /softwaremap
Disallow:  /top
Disallow:  /tracker
```

Figure A.6: An example robots.txt file (http://sourceforge.net/robots.txt).

In this section we present the various factors that crawlers must consider when crawling the Web in a conscientious and responsible manner.

Many of the problems relating to crawler etiquette relate to the differences in the behaviour of a typical web client and a crawler. The throughput of a crawler, for instance, is very different to that of a typical web client. A web browser typically sends a few request in a short period of time whenever a user clicks on a link. This is followed by a long period of inactivity, while the user reads the content of the returned documents, before they send their next request. In contrast, a crawler is an autonomous program that can send multiple, continuous, requests in parallel to many web servers.

The Web is an environment with limited resources, operating across networks of varying bandwidth, and so it is important that a crawler does not send a large number of requests over a short period of time, to any particular server. If a crawler floods a web server with requests, this will produce what is known as a Denial Of Service (DOS) attack and the web server will not be able to respond to requests from other web clients. This sort of behaviour will generally get the web crawler banned from the site, it is therefore important that the crawler restrict the frequency of requests to any particular domain.

Crawlers should also follow any robot exclusion guidelines that have been outlined by the web site administrator. These rules apply throughout the entire domain and are identified in

```
<HTML>
  <HEAD>
    <META name="ROBOTS" content="NOINDEX,NOFOLLOW">
  </HEAD>
  <BODY>
    <P> ... </P>
  </BODY>
</HTML>
```

Figure A.7: An example of robots META tags.

a file named "`robots.txt`" that is located in the root path of the site. The robot exclusion guidelines identify the resources that a crawler is not allowed to retrieve from the domain.

An example of the robot exclusion guidelines from the sourceforge.net domain are outlined in Figure A.6. The "`User-agent`" field identifies the web crawler. If a particular web crawler is not specifically identified, it must follow the rules outlined for "`User-agent: *`". This is followed by a list of resources that the identified crawler is prohibited from retrieving. For instance, the first rule "`Disallow: /forum`" indicates that any URL with the prefix "`www.slashdot.org/forum`" cannot be retrieved by any crawler, other than the user agent "`Mediapartners-Google*`", which has no restrictions.

While a web administrator can define domain wide crawler restrictions by creating a robots.txt file, web authors can create their own robots exclusion guidelines for individual HTML resources through the use of HTML META tags, as shown in Figure A.7. The META tag has two parameters that can be altered. The first parameter indicates whether the current resource can be indexed or not, while the second parameter indicates whether the URL links in the resource can be followed. In this particular example, the first parameter is set to "`NOINDEX`", informing the crawler that the resource should not be indexed. Alternatively, it could be set to "`INDEX`". The second parameter in this example is set to "`NOFOLLOW`", informing the crawler not to follow any links in the resource. Alternatively, it could be set to "`FOLLOW`", allowing links in the HTML resource to be traversed.

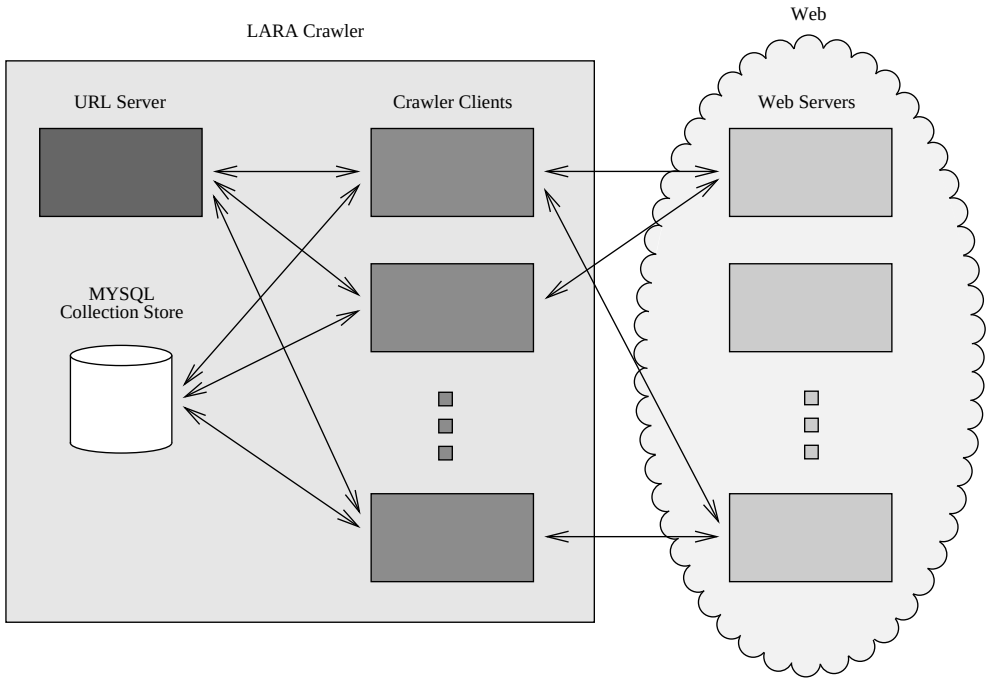


Figure A.8: Architecture of our web crawler.

By following these various guidelines a crawler can function with minimal impact on web servers and other web clients. In the next section we introduce the architecture and implementation of our LARA crawler.

A.8 Crawler Implementation

In this section we describe the various components of our LARA crawler and their design and implementation. This scalable crawler has been implemented in C code, compiled using the GNU C compiler [2006], and runs under the Linux/Unix environment. It uses a MySQL [2006] database as a backend for storing crawled data and has a distributed architecture. The system consists of three main components, outlined in Figure A.8, a MySQL database backend, a centralised URL server component, and distributed crawler components.

The number of distributed crawler components can be adjusted dynamically during runtime and can be run on different physical machines if required. The crawler components retrieve HTML documents from the Web and then extract and process their URLs.

Each extracted URL is compared to a URL filter and matching URLs are transmitted to the centralised URL server. The centralised server maintains the seen URL list, the list of resources to visit, and provides each crawler component with the next URL to retrieve. The seen URL list is maintained in a hashtable, while the list of resources to visit is maintained in a priority queue. The default crawl order is a breadth-first traversal.

The centralised URL server ensures that the same resource is not retrieved more than once by any crawler component, the distributed crawlers help improve scalability and throughput, while the database allows simple maintenance of the crawled data.

A.9 Summary

In this section we have presented some of the main challenges that we have been faced with during the development of a distributed web crawler and the solutions we have implemented. We have also discussed some of the details of our web crawler implementation. When collecting data for this thesis, we have used our crawler to trawl two large enterprise web sites with approximately 700,000 documents in total, ten times over a period of ten weeks, producing over 80 gigabytes of data.

Bibliography

H. Ali and H. E. Williams. What's Changed? Measuring Document Change in Web Crawling for Search Engines. In *String Processing and Information Retrieval (SPIRE)*, pages 28–42, Manaus, Brazil, 2003.

Alltheweb, Mar. 2008. URL http://www.alltheweb.com/.

Altavista, Mar. 2008. URL http://www.altavista.com/.

G. Amati, I. Ounis, and V. Plachouras. The Dynamic Absorbing Model for the Web. Technical Report TR-2003-137, University of Glasgow, Department of Computing Science, Apr. 2003.

E. Amitay. Using Common Hypertext Link to Identify the Best Phrasal Description of Target Web Documents. In *Proceedings of the ACM Special Interest Group on Information Retrieval (SIGIR 98) Post Conference Workshop on Hypertext Information Retrieval for the Web*. ACM Press, New York, New York, USA, 1998. URL http://www.mri.mq.edu.au/~einat/sigir/.

A. Arasu, J. Cho, H. García-Molina, A. Paepcke, and S. Raghavan. Searching the Web. *ACM Transactions on Internet Technology*, 1(1):2–43, Aug. 2001.

Ask.com, Mar. 2008. URL http://ask.com/.

R. A. Baeza-Yates. Challenges in the Interaction of Information Retrieval and Natural Language Processing. In A. F. Gelbukh, editor, *Computational Linguistics and Intelligent Text Processing, Fifth International Conference, CICLing 2004*, volume 2945 of *Lecture Notes in Computer Science*, pages 445–456, Seoul, Korea, Feb. 15–21 2004. Springer. ISBN

3-540-21006-7. URL http://springerlink.metapress.com/openurl.asp?genre=article&issn=0302-9743&volume=2945&spage=445.

R. A. Baeza-Yates and C. Castillo. Relating Web Characteristics with Link Based Web Page Ranking. In *Proceedings of String Processing and Information Retrieval (SPIRE)*, pages 21–32, Laguna de San Rafael, Chile, Nov. 13–15 2001. IEEE CS. Press.

R. A. Baeza-Yates and C. Castillo. Balancing Volume, Quality and Freshness in Web Crawling. In A. Abraham, J. Ruiz-delSolar, and M. Köppen, editors, *Hybrid Intelligent Systems*, volume 87 of *Frontiers in Artificial Intelligence and Applications*, pages 565–572, Santiago, Chile, Dec. 1–4 2002. IOS Press. ISBN 1-58603-297-6. URL http://citeseer.ist.psu.edu/baeza-yates02balancing.html.

R. A. Baeza-Yates and C. Castillo. Crawling the Infinite Web: Five Levels are Enough. In S. Leonardi, editor, *Algorithms and Models for the Web-Graph: Third International Workshop, (WAW 2004)*, volume 3243 of *Lecture Notes in Computer Science*, pages 156–167, Rome, Italy, Oct. 16 2004. Springer. ISBN 3-540-23427-6. URL http://springerlink.metapress.com/openurl.asp?genre=article&issn=0302-9743&volume=3243&spage=156.

R. A. Baeza-Yates and E. Davis. Web Page Ranking using Link Attributes. In S. I. Feldman, M. Uretsky, M. Najork, and C. E. Wills, editors, *WWW Alternate '04: Proceedings of the Thirteenth International World Wide Web Conference on Alternate Track Papers & Posters*, pages 328–329, New York, New York, USA, May 17–20 2004. ACM Press, New York, New York, USA. ISBN 1-58113-912-8. doi: http://doi.acm.org/10.1145/1013367.1013459.

R. A. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley and ACM Press, Harlow, United Kingdom, 1999. ISBN 0-201-39829-X. URL http://sunsite.dcc.uchile.cl/irbook/.

R. A. Baeza-Yates, C. Castillo, and F. Saint-Jean. Web Dynamics, Structure and Page Quality. In M. Levene and A. Poulovassilis, editors, *Web Dynamics*, pages 93–109. Springer Verlag, 2004.

R. A. Baeza-Yates, C. Castillo, M. Marin, and A. Rodriguez. Crawling a Country: Better Strategies than Breadth-First for Web Page Ordering. In A. Ellis and T. Hagino, editors,

WWW '05: Special Interest Tracks and Posters of the Fourteenth International Conference on World Wide Web, pages 864–872, Chiba, Japan, May 10–14 2005. ACM Press, New York, New York, USA. ISBN 1-59593-051-5. doi: http://doi.acm.org/10.1145/1062768.

Z. Bar-Yossef, A. Z. Broder, R. Kumar, and A. Tomkins. Sic Transit Gloria Telae: Towards an Understanding of the Web's Decay. In S. I. Feldman, M. Uretsky, M. Najork, and C. E. Wills, editors, *WWW '04: Proceedings of the Thirteenth International Conference on World Wide Web*, pages 328–337, New York, New York, USA, May 17–20 2004. ACM Press, New York, New York, USA. ISBN 1-58113-844-X. doi: http://doi.acm.org/10.1145/988672.988716.

A.-L. Barabási and R. Albert. Emergence of Scaling in Random Networks. *Science*, 286 (5439):509–512, Oct. 15 1999. ISSN 0036-8075.

D. Barbará and H. García-Molina. The Demarcation Protocol: A Technique for Maintaining Constraints in Distributed Database Systems. *VLDB Journal*, 3(3):325–353, 1994.

A. S. Z. Belloum and L. O. Hertzberger. Concurrent Evaluation of Web Cache Replacement and Coherence Strategies. *Simulation*, 78(1):28–35, Jan. 2002.

T. Berners-Lee. W3C, Mar. 2008. URL http://www.w3.org/.

T. Berners-Lee, R. Cailliau, J. Groff, and B. Pollermann. World-Wide Web: The Information Universe. *Electronic Networking: Research, Applications and Policy*, 1(2):74–82, 1992. URL http://citeseer.ist.psu.edu/berners-lee92worldwide.html.

P. A. Bernstein, B. T. Blaustein, and E. M. Clarke. Fast Maintenance of Semantic Integrity Assertions Using Redundant Aggregate Data. In *Sixth International Conference on Very Large Data Bases*, pages 126–136, Montreal, Canada, Oct. 1–3 1980. IEEE Computer Society.

K. Bharat and A. Broder. Mirror, Mirror on the Web: A Study of Host Pairs with Replicated Content. In P. H. Enslow Jr., editor, *WWW '99: Proceeding of the Eighth International Conference on World Wide Web*, pages 1579–1590, Toronto, Canada, May 11–14 1999. Elsevier North-Holland Inc., New York, New York, USA. doi: http://dx.doi.org/10.1016/S1389-1286(99)00021-3.

K. Bharat and M. R. Henzinger. Improved Algorithms for Topic Distillation in a Hyperlinked Environment. In *SIGIR '98: Proceedings of the Twenty-First Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 104–111, Melbourne, Australia, 1998. ACM Press, New York, New York, USA. ISBN 1-58113-015-5. doi: http://doi.acm.org/10.1145/290941.290972.

K. Bharat and G. A. Mihaila. Hilltop: A Search Engine Based on Expert Documents. Technical Report CSRG-405, University of Toronto, Feb. 2000. URL http://www.cs.toronto.edu/~georgem/hilltop/. ftp://ftp.cs.toronto.edu/csrg-technical-reports/405/hilltop.html.

P. Boldi, B. Codenotti, M. Santini, and S. Vigna. UbiCrawler: A Scalable Fully Distributed Web Crawler. *Software – Practice and Experience*, 34(8):711–726, 2004a. ISSN 0038-0644. doi: http://dx.doi.org/10.1002/spe.587.

P. Boldi, M. Santini, and S. Vigna. Do Your Worst to Make the Best: Paradoxical Effects in PageRank Incremental Computations. In S. Leonardi, editor, *Algorithms and Models for the Web-Graph: Third International Workshop, (WAW 2004)*, volume 3243 of *Lecture Notes in Computer Science*, pages 168–180, Rome, Italy, Oct. 16 2004b. Springer. ISBN 3-540-23427-6. URL http://springerlink.metapress.com/openurl.asp?genre=article&issn=0302-9743&volume=3243&spage=168.

V. Boyapati, K. Chevrier, A. Finkel, N. Glance, T. Pierce, R. Stockton, and C. Whitmer. ChangeDetector$^{\mathrm{TM}}$: A Site Level Monitoring Tool for the WWW. In *WWW '02: Proceedings of the Eleventh International Conference on World Wide Web*, pages 570–579, Honolulu, Hawaii, USA, May 7–11 2002. ACM Press, New York, New York, USA. ISBN 1-58113-449-5. doi: http://doi.acm.org/10.1145/511446.511464.

O. Brandman, J. Cho, H. García-Molina, and N. Shivakumar. Crawler-Friendly Web Servers. *SIGMETRICS Performance Evaluation Revision*, 28(2):9–14, 2000. ISSN 0163-5999. doi: http://doi.acm.org/10.1145/362883/362894.

B. E. Brewington and G. Cybenko. Keeping Up with the Changing Web. *Computer*, 33(5): 52–58, 2000a. ISSN 0018-9162. doi: http://dx.doi.org/10.1109/2.841784.

B. E. Brewington and G. Cybenko. How Dynamic is the Web? In *Proceedings of the Ninth International World Wide Web Conference on Computer Networks: The International*

Journal of Computer and Telecommunications Networking, pages 257–276, Amsterdam, The Netherlands, 2000b. North-Holland Publishing Co., Amsterdam, The Netherlands. doi: http://dx.doi.org/10.1016/S1389-1286(00)00045-1.

S. Brin and L. Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. In P. H. Enslow Jr. and A. Ellis, editors, *WWW7: Proceedings of the Seventh International Conference on World Wide Web*, pages 107–117, Brisbane, Australia, Apr. 14–18 1998. Elsevier Science Publishers B. V., Amsterdam, The Netherlands. doi: http://dx.doi.org/10.1016/S0169-7552(98)00110-X.

A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener. Graph Structure in the Web. In *Proceedings of the Ninth International World Wide Web Conference on Computer Networks : The International Journal of Computer and Telecommunications Networking*, pages 309–320, Amsterdam, The Netherlands, 2000a. North-Holland Publishing Co., Amsterdam, The Netherlands. doi: http://dx.doi.org/10.1016/S1389-1286(00)00083-9.

A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener. Graph Structure in the Web. *Computer Networks*, 33(1-6):309–320, June 2000b. ISSN 1389-1286. doi: http://dx.doi.org/10.1016/S1389-1286(00)00083-9.

A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic Clustering of the Web. *Computer Networks and ISDN Systems*, 29(8-13):1157–1166, 1997. ISSN 0169-7552. doi: http://dx.doi.org/10.1016/S0169-7552(97)00031-7.

A. Z. Broder, M. Najork, and J. L. Wiener. Efficient URL Caching for World Wide Web Crawling. In *WWW '03: Proceedings of the Twelfth International Conference on World Wide Web*, pages 679–689, Budapest, Hungary, May 20–24 2003. ACM Press, New York, New York, USA. ISBN 1-58113-680-3. doi: http://doi.acm.org/10.1145/775152.775247.

A. Z. Broder, R. Lempel, F. Maghoul, and J. O. Pedersen. Efficient PageRank Approximation via Graph Aggregation. In S. I. Feldman, M. Uretsky, M. Najork, and C. E. Wills, editors, *WWW Alt. '04: Proceedings of the Thirteenth International World Wide Web Conference on Alternate Track Papers & Posters*, pages 484–485, New York, New York, USA, May 17–

20 2004. ACM Press, New York, New York, USA. ISBN 1-58113-912-8. doi: http://doi.acm.org/10.1145/1013367.1013537.

A. Z. Broder, R. Lempel, F. Maghoul, and J. O. Pedersen. Efficient PageRank Approximation via Graph Aggregation. *Information Retrieval*, 9(2):123–138, 2006. doi: http://dx.doi.org/10.1007/s10791-006-7146-1.

T. A. Brooks. Web Search: How the Web has Changed Information Retrieval. *Information Research*, 8(3), 2003. URL http://informationr.net/ir/8-3/paper154.html.

H. Bullot, S. K. Gupta, and M. K. Mohania. A Data-Mining Approach for Optimizing Performance of an Incremental Crawler. In *2003 IEEE / WIC International Conference on Web Intelligence, (WI 2003)*, pages 610–615, Halifax, Canada, Oct. 13–17 2003. IEEE Computer Society, Los Alamitos, California, USA. ISBN 0-7695-1932-6. doi: http://doi.ieeecomputersociety.org/10.1109/WI.2003.1241279. URL http://csdl.computer.org/comp/proceedings/wi/2003/1932/00/19320610abs.htm1023.

M. Burner. Crawling Towards Eternity - Building an Archive of the World Wide Web. *Web Techniques*, 2(5), May 1997.

J. P. Callan and M. E. Connell. Query-Based Sampling of Text Databases. *ACM Transactions on Information Systems*, 19(2):97–130, 2001. ISSN 1046-8188. doi: http://doi.acm.org/10.1145/382979.383040.

P. Cao and C. Liu. Maintaining Strong Cache Consistency in the World Wide Web. *IEEE Transactions on Computers*, 47(4):445–457, 1998. ISSN 0018-9340. doi: http://dx.doi.org/10.1109/12.675713.

S. J. Carrière and R. Kazman. WebQuery: Searching and Visualizing the Web through Connectivity. *Computer Networks*, 29(8-13):1257–1267, 1997. doi: http://dx.doi.org/10.1016/S0169-7552(97)00062-7.

O. S. F. Carvalho and G. Roucairol. On the Distribution of an Assertion. In *PODC '82: Proceedings of the First ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, pages 121–131, Ottawa, Canada, Aug. 18–20 1982. ACM Press, New York, New York, USA. ISBN 0-89791-081-8. doi: http://doi.acm.org/10.1145/800220.806689.

C. Castillo. *Effective Web Crawling*. PhD thesis, University of Chile, Nov. 2004.

V. Cate. Alex – A Global File System. In *Proceedings of the 1992 USENIX File System Workshop*, pages 1–11, Ann Arbor, Michigan, USA, May 21–22 1992. URL http://citeseer.ist.psu.edu/cate92alex.html.

S. Chakrabarti. *Mining the Web: Discovering Knowledge from Hypertext Data*. Morgan-Kauffman, 2002. ISBN 1-55860-754-4. URL http://www.cse.iitb.ac.in/~soumen/mining-the-web/.

S. Chakrabarti, B. Dom, P. Raghavan, S. Rajagopalan, D. Gibson, and J. Kleinberg. Automatic Resource Compilation by Analyzing Hyperlink Structure and Associated Text. In P. H. Enslow Jr. and A. Ellis, editors, *WWW7: Proceedings of the Seventh International Conference on World Wide Web*, pages 65–74, Brisbane, Australia, Apr. 14–18 1998. Elsevier Science Publishers B. V., Amsterdam, The Netherlands. doi: http://dx.doi.org/10.1016/S0169-7552(98)00087-7.

S. Chakrabarti, M. V. D. Berg, and B. Dom. Focused Crawling: A New Approach to Topic-Specific Web Resource Discovery. *Computer Networks*, 31(11–16):1623–1640, 1999. ISSN 1389-1286. URL http://citeseer.ist.psu.edu/chakrabarti99focused.html.

S. Chakrabarti, K. Punera, and M. Subramanyam. Accelerated Focused Crawling through Online Relevance Feedback. In *WWW '02: Proceedings of the Eleventh International Conference on World Wide Web*, pages 148–159, Honolulu, Hawaii, USA, May 7–11 2002. ACM Press, New York, New York, USA. ISBN 1-58113-449-5. doi: http://doi.acm.org/10.1145/511446.511466.

A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell. A Hierarchical Internet Object Cache. In *USENIX Annual Technical Conference*, pages 153–164, San Diego, California, USA, Jan. 22–26 1996. USENIX Association. URL http://citeseer.ist.psu.edu/chankhunthod95hierarchical.html.

M. Chau and H. Chen. Comparison of Three Vertical Search Spiders. *IEEE Computer*, 36 (5):56–62, 2003. URL http://computer.org/computer/co2003/r5056abs.htm.

S. S. Chawathe and H. García-Molina. Meaningful Change Detection in Structured Data. In J. Peckham, editor, *SIGMOD '97, Proceedings ACM SIGMOD International Conference on Management of Data*, pages 26–37, Tucson, Arizona, USA, May 13–15 1997. ACM Press, New York, New York, USA. ISBN 0-89791-911-4. doi: http://doi.acm.org/10.1145/253260.253266.

K.-J. Chen and S.-H. Liu. Word Identification for Mandarin Chinese Sentences. In *COLING 1992, Proceedings of the Fourteenth International Conference on Computational Linguistics*, pages 101–107, Nantes, France, Aug. 23–28 1992. Association for Computational Linguistics, Morristown, New Jersey, USA. URL http://acl.ldc.upenn.edu/C/C92/C92-1019.pdf.

Y.-Y. Chen, Q. Gan, and T. Suel. I/O-Efficient Techniques for Computing PageRank. In *CIKM '02: Proceedings of the Eleventh International Conference on Information and Knowledge Management*, pages 549–557, McLean, Virginia, USA, Nov. 4–9 2002. ACM Press, New York, New York, USA. ISBN 1-58113-492-4. doi: http://doi.acm.org/10.1145/584792.584882.

J. Cho. *Crawling the Web: Discovery and Maintenance of Large-Scale Web Data*. PhD thesis, Stanford University, 2001. URL http://citeseer.ist.psu.edu/cho01crawling.html.

J. Cho and H. García-Molina. Synchronizing a Database to Improve Freshness. In *SIGMOD '00: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 117–128, Dallas, Texas, USA, 2000a. ACM Press, New York, New York, USA. ISBN 1-58113-217-4. doi: http://doi.acm.org/10.1145/342009.335391.

J. Cho and H. García-Molina. Estimating Frequency of Change. Technical report, Stanford University, Computer Science Department, Nov. 2000b.

J. Cho and H. García-Molina. The Evolution of the Web and Implications for an Incremental Crawler. In *VLDB '00: Proceedings of the Twenty Sixth International Conference on Very Large Data Bases*, pages 200–209, Cairo, Egypt, Sept. 10–14 2000c. Morgan Kaufmann Publishers Inc., San Francisco, California, USA. ISBN 1-55860-715-3.

J. Cho and H. García-Molina. Parallel Crawlers. In *Proceedings of the Eleventh International Conference on World Wide Web*, pages 124–135, Honolulu, Hawaii, USA, May 7–11 2002.

ACM Press, New York, New York, USA. ISBN 1-58113-449-5. doi: http://doi.acm.org/10.1145/511446.511464.

J. Cho and H. García-Molina. Estimating Frequency of Change. *ACM Transactions on Internet Technology (TOIT)*, 3(3):256–290, 2003a. ISSN 1533-5399. doi: http://doi.acm.org/10.1145/857166.857170.

J. Cho and H. García-Molina. Effective Page Refresh Policies for Web Crawlers. *ACM Transactions on Database Systems (TODS)*, 28(4):390–426, 2003b. ISSN 0362-5915. doi: http://doi.cm.org/10.1145/958942.958945.

J. Cho and A. Ntoulas. Effective Change Detection Using Sampling. In *VLDB 2002, Proceedings of Twenty Eighth International Conference on Very Large Data Bases*, pages 514–525, Hong Kong, China, Aug. 2002. Morgan Kaufmann. URL http://www.vldb.org/conf/2002/S15P01.pdf.

J. Cho, H. García-Molina, and L. Page. Efficient Crawling through URL Ordering. *Computer Networks and ISDN Systems*, 30(1–7):161–172, 1998. URL http://citeseer.ist.psu.edu/cho98efficient.html.

J. Cho, N. Shivakumar, and H. García-Molina. Finding Replicated Web Collections. In *SIGMOD '00: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 355–366, Dallas, Texas, USA, 2000. ACM Press, New York, New York, USA. ISBN 1-58113-217-4. doi: http://doi.acm.org/10.1145/342009.335429.

E. G. Coffman, Z. Liu, and R. R. Weber. Optimal Robot Scheduling for Web Search Engines. *Journal of Scheduling*, 1:15–29, 1998.

Comscore, Mar. 2004. URL http://www.comscore.com/press/release.asp?press=458.

Corporate Adobe Systems Inc. PDF Reference, Dec. 1993. URL http://partners.adobe.com/public/developer/pdf/index_reference.html.

Corporate Adobe Systems Inc. *PostScript Language Reference*. Addison-Wesley Longman Publishing Co. Inc., Boston, Massachusetts, USA, third edition, 1999. ISBN 0-201-37922-8. URL http://www.adobe.com/products/postscript/pdfs/PLRM.pdf.

N. Craswell, D. Hawking, and S. Robertson. Effective Site Finding using Link Anchor Information. In *SIGIR '01: Proceedings of the Twenty Fourth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 250–257, New Orleans, Louisiana, USA, Sept. 9–13 2001. ACM Press, New York, New York, USA. ISBN 1-58113-331-6. doi: http://doi.acm.org/10.1145/383952.383999.

A. Czumaj, I. Finch, L. Gasieniec, A. Gibbons, P. Leng, W. Rytter, and M. Zito. Efficient Web Searching using Temporal Factors. *Theoretical Computer Science*, 262(1–2):569–582, 2001. ISSN 0304-3975. doi: http://dx.doi.org/10.1016/S0304-3975(00)00366-2.

A. S. da Silva, E. A. Veloso, P. B. Golghe, B. Ribeiro-Neto, A. H. F. Laender, and N. Ziviani. CoBWeb: A Crawler for the Brazilian Web. In *SPIRE '99: Proceedings of the String Processing and Information Retrieval Symposium & International Workshop on Groupware*, page 184. IEEE Computer Society, Washington DC, Washington, USA, 1999. ISBN 0-7695-0268-7.

M. Dasen and E. Wilde. Keeping Web Indices Up-To-Date. In *Poster Proceedings of the Tenth International World Wide Web Conference, WWW 10*, Hong Kong, China, May 1–5 2001. ACM Press, New York, New York, USA. URL http://www10.org/cdrom/posters/1132.pdf.

DASL. DAV Searching and Locating Protocol, Apr. 2000. URL http://www.webdav.org/dasl/.

B. D. Davison. Topical Locality in the Web. In N. J. Belkin, P. Ingwersen, and M. K. Leong, editors, *SIGIR '00: Proceedings of the Twenty Third Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 272–279, Athens, Greece, July 24–28 2000. ACM Press, New York, New York, USA. ISBN 1-58113-226-3. doi: http://doi.acm.org/10.1145/345508.345597.

P. M. E. De Bra and R. D. J. Post. Information Retrieval in the World-Wide Web: Making Client-Based Searching Feasible. *Computer Networks and ISDN Systems*, 27(2):183–192, 1994. URL http://citeseer.ist.psu.edu/99604.html.

P. J. Deutsch. Original Archie Announcement, 1990. URL http://groups.google.com/group/comp.archives/msg/a77343f9175b24c3?output=gplain.

F. Diaz and R. Jones. Using Temporal Profiles of Queries for Precision Prediction. In *SIGIR '04: Proceedings of the Twenty Seventh Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 18–24, Sheffield, United Kingdom, 2004. ACM Press, New York, New York, USA. ISBN 1-58113-881-4. doi: http://doi.acm.org/10.1145/1008992.1008998.

M. Diligenti, F. Coetzee, S. Lawrence, C. L. Giles, and M. Gori. Focused Crawling using Context Graphs. In *VLDB '00: Proceedings of the Twenty Sixth International Conference on Very Large Data Bases*, pages 527–534, Cairo, Egypt, Sept. 10–14 2000. Morgan Kaufmann Publishers Inc., San Francisco, California, USA. ISBN 1-55860-715-3. URL http://citeseer.ist.psu.edu/diligenti00focused.html.

M. Diligenti, M. Gori, and M. Maggini. A Unified Probabilistic Framework for Web Page Scoring Systems. *IEEE Transactions on Knowledge and Data Engineering*, 16(1):4–16, 2004. ISSN 1041-4347. doi: http://dx.doi.org/10.1109/TKDE.2004.1264818.

S. Dill, R. Kumar, K. S. Mccurley, S. Rajagopalan, D. Sivakumar, and A. Tomkins. Self-Similarity in the Web. *ACM Transactions on Internet Technology (TOIT)*, 2(3):205–223, Aug. 2002. ISSN 1533-5399. doi: http://doi.acm.org/10.1145/572326.572328.

DMOZ. The Open Directory Project, Mar. 2008. URL http://www.dmoz.org/.

Dogpile, Mar. 2008. URL http://www.dogpile.com/.

F. Douglis, A. Feldmann, B. Krishnamurthy, and J. C. Mogul. Rate of Change and Other Metrics: A Live Study of the World Wide Web. In *USENIX Symposium on Internet Technologies and Systems*, Dec. 1997. URL http://citeseer.ist.psu.edu/douglis97rate.html.

J. Edwards, K. S. McCurley, and J. A. Tomlin. An Adaptive Model for Optimizing Performance of an Incremental Web Crawler. In *WWW '01: Proceedings of the Tenth International World Wide Web Conference*, pages 106–113, Hong Kong, China, May 1–5 2001. ACM Press, New York, New York, USA. ISBN 1-58113-348-0. doi: http://doi.acm.org/10.1145/371920.371960.

D. Eichmann. The RBSE Spider - Balancing Effective Search Against Web Load. *Computer Networks and ISDN Systems*, May 1994. URL http://citeseer.ist.psu.edu/eichmann94rbse.html.

N. Eiron and K. S. McCurley. Untangling compound documents on the web. In *HYPERTEXT '03: Proceedings of the Fourteenth ACM Conference on Hypertext and Hypermedia*, pages 85–94, Nottingham, United Kingdom, Aug. 26–30 2003a. ACM Press, New York, New York, USA. ISBN 1-58113-704-4. doi: http://doi.acm.org/10.1145/900051.900070.

N. Eiron and K. S. McCurley. Locality, Hierarchy, and Bidirectionality in the Web. In *Workshop on Algorithms and Models for the Web Graph*, Budapest, Hungary, May 20 2003b. URL http://citeseer.ist.psu.edu/eiron03locality.html.

N. Eiron, K. S. McCurley, and J. A. Tomlin. Ranking the Web Frontier. In S. I. Feldman, M. Uretsky, M. Najork, and C. E. Wills, editors, *WWW '04: Proceedings of the Thirteenth International Conference on World Wide Web*, pages 309–318, New York, New York, USA, May 17–20 2004. ACM Press, New York, New York, USA. ISBN 1-58113-844-X. doi: http://doi.acm.org/10.1145/988672.988714.

A. Emtage and P. Deutsch. Archie: An Electronic Directory Service for the Internet. In *Proceedings of the Winter 1992 USENIX Conference*, pages 93–110, San Francisco, California, USA, 1991.

Excite, Mar. 2008. URL http://www.excite.com/.

R. Fagin, R. Kumar, K. S. McCurley, J. Novak, D. Sivakumar, J. A. Tomlin, and D. P. Williamson. Searching the Workplace Web. In *WWW '03: Proceedings of the Twelfth International Conference on World Wide Web*, pages 366–375, Budapest, Hungary, May 20–24 2003a. ACM Press, New York, New York, USA. ISBN 1-58113-680-3. doi: http://doi.acm.org/10.1145/775152.775204.

R. Fagin, R. Kumar, and D. Sivakumar. Comparing Top k Lists. In *SODA '03: Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 28–36. Society for Industrial and Applied Mathematics, Philadelphia, PA, Jan. 12–14 2003b. ISBN 0-89871-538-5.

D. Fetterly, M. Manasse, and M. Najork. On the Evolution of Clusters of Near-Duplicate Web Pages. In *LA-WEB '03: Proceedings of the First Conference on Latin American Web Congress, Empowering Our Web*, pages 37–45, Sanitago, Chile, Nov. 10–12 2003a. IEEE Computer Society, Washington DC, Washington, USA. ISBN 0-7695-2058-8. URL http://csdl.computer.org/comp/proceedings/la-web/2003/2058/00/20580037abs.htm.

D. Fetterly, M. Manasse, M. Najork, and J. L. Wiener. A Large-Scale Study of the Evolution of Web Pages. In *WWW '03: Proceedings of the Twelfth International Conference on World Wide Web*, pages 669–678, Budapest, Hungary, May 20–24 2003b. ACM Press, New York, New York, USA. ISBN 1-58113-680-3. doi: http://doi.acm.org/10.1145/775152.775246.

R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol: HTTP/1.1. Network Working Group: RFC 2616, June 1999a. URL http://www.w3.org/Protocols/rfc2616/rfc2616.html.

R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol: HTTP/1.1 – Section 14.21 Expires. Network Working Group: RFC 2616, June 1999b. URL http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.21.

Fireball, Mar. 2008. URL http://www.fireball.de/.

S. Flesca and E. Masciari. Efficient and Effective Web Change Detection. *Data & Knowledge Engineering*, 46(2):203–224, 2003. ISSN 0169-023X. doi: http://dx.doi.org/10.1016/S0169-023X(02)00210-0.

W. B. Frakes. Stemming Algorithms. In W. B. Frakes and R. A. Baeza-Yates, editors, *Information Retrieval: Data Structures & Algorithms*, pages 131–160. Prentice-Hall, 1992. ISBN 0-13-463837-9.

N. Fuhr, M. Lalmas, S. Malik, and Z. Szlavik, editors. *Advances in XML Information Retrieval: Third International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX 2004*, volume 3493 of *Lecture Notes in Computer Science*, Dagstuhl Castle, Germany, Dec. 6–8 2004. Springer-Verlag GmbH. ISBN 978-3-540-26166-7. URL http://www.springeronline.com/3-540-26166-4.

S. Garcia, H. E. Williams, and A. Cannane. Access-Ordered Indexes. In V. Estivill-Castro, editor, *ACSC '04: Proceedings of the Twenty-Seventh Australasian Computer Science Conference*, volume 26 of *Conferences in Research and Practice in Information Technology*, pages 7–14, Dunedin, New Zealand, Jan. 2004. Australian Computer Society Inc., Darlinghurst, Australia. ISBN 1-920682-05-8. URL http://crpit.com/confpapers/CRPITV26Garcia.ps.

M. Ghodsi, O. Hassanzadeh, S. Kamali, and M. Monenizadeh. A Hybrid Approach for Refreshing Web Page Repositories. In L. Zhou, B. C. Ooi, and X. Meng, editors, *DASFAA 2005: Proceedings of the Tenth International Conference on Database Systems for Advanced Applications*, volume 3453 of *Lecture Notes in Computer Science*, pages 588–593, Beijing, China, Apr. 2005. Springer. ISBN 3-540-25334-3.

GNU C compiler, Sept. 2006. URL http://gcc.gnu.org/.

R. A. Golding and D. D. E. Long. Modeling Replica Divergence in a Weak-Consistency Protocol for Global Scale Distribution Data Bases. Technical Report UCSC-CRL-93-09, University of California, Santa Cruz, California, USA, 1993. URL http://citeseer.ifi.unizh.ch/golding93modeling.html.

Google, Mar. 2008. URL http://www.google.com/.

Google. Google's New GoogleScout Feature Expands Scope of Search on the Internet, Sept. 1999. URL http://www.google.com/press/pressrel/pressrelease4.html.

Google Zeitgeist, Mar. 2008. URL http://www.google.com/intl/en/press/zeitgeist.html.

Googlerank.com. Google Dance: The Google's Update, 2005. URL http://www.googlerank.com/ranking/Ebook/dance.html.

L. Gravano, K. C.-C. Chang, H. García-Molina, and A. Paepcke. STARTS: Stanford Proposal for Internet Meta-Searching. In J. Peckham, editor, *SIGMOD '97: Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, pages 207–218, Tucson, Arizona, USA, May 13–15 1997. ACM Press, New York, New York, USA. ISBN 0-89791-911-4. doi: http://doi.acm.org/10.1145/253260.253299.

M. K. Gray. Web Growth Summary, 1996a. URL http://www.mit.edu/~mkgray/net/web-growth-summary.html.

M. K. Gray. World Wide Web Wanderer, 1996b. URL http://www.mit.edu/people/mkgray/net/.

D. A. Grossman and O. Frieder. *Information Retrieval: Algorithms and Heuristics*. Kluwer Academic Publishers, Norwell, Massachusetts, USA, 1998. ISBN 0792382714.

A. Gulli and A. Signorini. The Indexable Web is more than 11.5 Billion Pages. In A. Ellis and T. Hagino, editors, *WWW '05: Special Interest Tracks and Posters of the Fourteenth International Conference on World Wide Web*, pages 902–903, Chiba, Japan, May 10–14 2005. ACM Press, New York, New York, USA. ISBN 1-59593-051-5. doi: http://doi.acm.org/10.1145/1062745.1062789.

V. Gupta and R. Campbell. Internet Search Engine Freshness by Web Server Help. In *SAINT '01: Proceedings of the 2001 Symposium on Applications and the Internet (SAINT 2001)*, page 113, San Diego, California, USA, Jan. 8–12 2001. IEEE Computer Society, Washington DC, Washington, USA. ISBN 0-7695-0942-8.

J. Gwertzman and M. I. Seltzer. World Wide Web Cache Consistency. In *Proceedings of the USENIX Annual Technical Conference*, pages 141–152, San Diego, California, USA, Jan. 22–26 1996. USENIX Association. URL http://citeseer.ist.psu.edu/chankhunthod95hierarchical.html.

Y. Hafri and C. Djeraba. High Performance Crawling System. In *MIR '04: Proceedings of the Sixth ACM SIGMM International Workshop on Multimedia Information Retrieval*, pages 299–306, New York, New York, USA, 2004. ACM Press, New York, New York, USA. ISBN 1-58113-940-3. doi: http://doi.acm.org/10.1145/1026711.1026760.

H. Hahn and R. Stout. The Gopher, Veronica, and Jughead. In *The Internet Complete Reference*, pages 429–457. Osborne McGraw-Hill, 1994.

D. Harman. Overview of the Second Text REtrieval Conference (TREC-2). *Information Processing and Management*, 31(3):271–289, May/June 1995. doi: http://dx.doi.org/10.1016/0306-4573(94)00047-7.

J. Harris. Mining the Internet: Networked Information Location Tools: Gophers, Veronica, Archie, and Jughead. *Computing Teacher*, 21(1):16–19, Aug. 1 1993. ISSN 0278-9175.

T. H. Haveliwala. Topic-Sensitive PageRank. In *WWW '02: Proceedings of the Eleventh International Conference on World Wide Web*, pages 517–526, Honolulu, Hawaii, USA, May 7–11 2002. ACM Press, New York, New York, USA. ISBN 1-58113-449-5. doi: http://doi.acm.org/10.1145/511446.511513.

T. H. Haveliwala. Efficient Computation of PageRank. Technical Report 1999-31, Computer Science Department, Stanford University, 1999. URL http://citeseer.ist.psu.edu/haveliwala99efficient.html.

D. Hawking. Challenges in Enterprise Search. In *CRPIT '04: Proceedings of the Fifteenth Conference on Australasian Database*, pages 15–24, Dunedin, New Zealand, 2004. Australian Computer Society Inc., Darlinghurst, Australia.

D. Hawking and N. Craswell. Overview of the TREC-2001 Web Track. In E. M. Voorhees and D. K. Harman, editors, *Proceedings of the Tenth Text REtrieval Conference TREC-2001, NIST Special Publication 500-250*, pages 61–67, Gaithersburg, Maryland, USA, 2002. National Institute of Standards and Technology (NIST), Washington DC, Washington, USA.

D. Hawking and P. Thomas. Server Selection Methods in Hybrid Portal Search. In R. A. Baeza-Yates, N. Ziviani, G. Marchionini, A. Moffat, and J. Tait, editors, *SIGIR '05: Proceedings of the Twenty-Eighth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 75–82, Salvador, Brazil, Aug. 15–19 2005. ACM Press, New York, New York, USA. ISBN 1-59593-034-5. doi: http://doi.acm.org/10.1145/1076034.1076050.

D. Hawking, N. Craswell, and P. B. Thistlewaite. Overview of TREC-7 Very Large Collection Track. In *TREC-7, NIST Special Publication 500-242*, pages 40–52, Gaithersburg, Maryland, USA, 1998. National Institute of Standards and Technology (NIST), Washington DC, Washington, USA.

P. L. Hégaret, R. Whitmer, and L. Wood. W3C Document Object Model (DOM), June 12 2005. URL http://www.w3.org/DOM/.

M. R. Henzinger. Hyperlink Analysis for the Web. *IEEE Internet Computing*, 5(1):45–50, 2001. ISSN 1089-7801. doi: http://dx.doi.org/10.1109/4236.895141. URL http://www.computer.org/internet/ic2001/w1045abs.htm.

A. Heydon and M. Najork. Mercator: A Scalable, Extensible Web Crawler. *World Wide Web*, 2(4):219–229, 1999. ISSN 1386-145X. doi: http://dx.doi.org/10.1023/A:1019213109274. URL http://citeseer.ist.psu.edu/heydon99mercator.html.

T. C. Hoad and J. Zobel. Methods for Identifying Versioned and Plagiarized Documents. *Journal of the American Society for Information Science and Technology (JASIST)*, 54 (3):203–215, 2003. ISSN 1532-2882. doi: http://dx.doi.org/10.1002/asi.10170.

B. A. Huberman and L. A. Adamic. Evolutionary Dynamics of the World Wide Web. *Nature*, 401:131, 1999. URL http://www.citebase.org/cgi-bin/citations?id=oai:arXiv.org:cond-mat/9901071.

Infoseek, Mar. 2008. URL http://www.infoseek.co.jp/.

Inktomi, Mar. 2008. URL http://www.inktomi.com/.

Internet Archive, Sept. 2006. URL http://web.archive.org/.

Internet Archive. Internet Archive Wayback Machine, Mar. 2008. URL http://web.archive.org/web/*/http://www.rmit.edu.au.

iProspect. iProspect Search Engine User Behavior Study, Apr. 2006. URL http://www.iprospect.com/premiumPDFs/WhitePaper_2006_SearchEngineUserBehavior.pdf.

ISC. Internet Software Consortium, Internet Domain Survey, 2006. URL http://www.ics.org.

B. J. Jansen, A. Spink, J. Bateman, and T. Saracevic. Real Life Information Retrieval: A Study of User Queries on the Web. *SIGIR Forum*, 32(1):5–17, 1998. ISSN 0163-5840. doi: http://doi.acm.org/10.1145/281250.281253.

T. Joachims. Optimizing Search Engines using Clickthrough Data. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 133–142, Edmonton, Canada, July 23–26 2002. ACM Press. ISBN 1-58113-567-X. doi: http://doi.acm.org/10.1145/775047.775067.

K. S. Jones, S. Walker, and S. E. Robertson. A Probabilistic Model of Information Retrieval: Development and Comparative Experiments - Part 1 and 2. *Information Processing and Management*, 36(6):779–840, 2000. URL http://citeseer.ist.psu.edu/sparckjones98probabilistic.html.

R. Jones. Jughead: Jonzy's Universal Gopher Hierarchy Excavation And Display. unpublished, Apr. 1993.

S. Kamvar, T. Haveliwala, and G. Golub. Adaptive Methods for the Computation of Pagerank. Technical report, Computer Science Department, Stanford University, Apr. 2003a. URL http://citeseer.ist.psu.edu/kamvar03adaptive.html.

S. D. Kamvar, T. H. Haveliwala, C. D. Manning, and G. H. Golub. Extrapolation Methods for Accelerating PageRank Computations. In *WWW '03: Proceedings of the Twelfth International Conference on World Wide Web*, pages 261–270, Budapest, Hungary, May 20–24 2003b. ACM Press, New York, New York, USA. ISBN 1-58113-680-3. doi: http://doi.acm.org/10.1145/775152.775190.

M. G. Kendall and J. D. Gibbons. *Rank Correlation Methods*. Oxford University Press, New York, New York, USA, fifth edition, 1990. ISBN 0195208374.

J. M. Kleinberg. Authoritative Sources in a Hyperlinked Environment. *Journal of the ACM*, 46(5):604–632, 1999. ISSN 0004-5411. doi: http://doi.acm.org/10.1145/324133.324140.

W. Koehler. Web Page Change and Persistence - A Four-Year Longitudinal Study. *Journal of the American Society for Information Science and Technology (JASIST)*, 53(2):162–171, 2002. ISSN 1532-2882. doi: http://dx.doi.org/10.1002/asi.10018.

W. Koehler. A Longitudinal Study of Web Pages Continued: A Consideration of Document Persistence. *Information Research*, 9(2), Jan. 2004.

M. Koster. Guidelines for Robots Writers, 1993. URL http://www.robotstxt.org/wc/guidelines.html.

M. Koster. ALIWEB - Archie-Like Indexing in the WEB. *Computer Networks and ISDN Systems*, 27(2):175–182, 1994a. ISSN 0169-7552. doi: http://dx.doi.org/10.1016/0169-7552(94)90131-7.

M. Koster. Robots in the Web: Threat or Treat? *ConneXions*, 9(4), Apr. 1995. URL http://www.robotstxt.org/wc/threat-or-treat.html.

M. Koster. A Standard for Robot Exclusion, 1994b. URL http://www.robotstxt.org/wc/norobots.html. http://www.robotstxt.org/wc/exclusion.html.

W. Kraaij, P. Over, and A. F. Smeaton. TREVID 2006 - An Introduction. In *Proceedings of the TREC Video Retrieval Evaluation (TRECVID) Workshop 2006*, Gaithersburg, Maryland, USA, Nov. 13–16 2006. National Institute of Standards and Technology (NIST), Washington DC, Washington, USA. URL http://www-nlpir.nist.gov/projects/tvpubs/tv.pubs.org.html.

N. Krishnakumar and A. J. Bernstein. Bounded Ignorance in Replicated Systems. In *PODS '91: Proceedings of the Tenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 63–74, Denver, Colorado, USA, May 29–31 1991. ACM Press, New York, New York, USA. ISBN 0-89791-430-9. doi: http://doi.acm.org/10.1145/113413.113419.

R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. Trawling the Web for Emerging Cyber-Communities. In P. H. Enslow Jr., editor, *WWW '99: Proceeding of the Eighth International Conference on World Wide Web*, pages 1481–1493, Toronto, Canada, May 11–14 1999. Elsevier North-Holland Inc., New York, New York, USA. doi: http://dx.doi.org/10.1016/S1389-1286(99)00040-7.

S. Lawrence and C. L. Giles. Searching the World Wide Web. *Science*, 280(5360):98–100, 1998. URL http://citeseer.ist.psu.edu/lawrence98searching.html.

S. Lawrence and C. L. Giles. Accessibility of Information on the Web. *Intelligence*, 11(1): 32–39, 2000. ISSN 1523-8822. doi: http://doi.acm.org/10.1145/333175.333181.

R. Lempel and S. Moran. Predictive Caching and Prefetching of Query Results in Search Engines. In *WWW '03: Proceedings of the Twelfth International Conference on World Wide Web*, pages 19–28, Budapest, Hungary, May 20–24 2003. ACM Press, New York, New York, USA. ISBN 1-58113-680-3. doi: http://doi.acm.org/10.1145/775152.775156.

N. Lester. *Efficient Index Maintenance for Text Databases*. PhD thesis, RMIT University, Aug. 2006.

D. Lewandowski. Date Restricted Queries in Web Search Engines. *Online Information Review*, 28(6):420–427, June 1 2004. ISSN 1468-4527. URL http://www.citebase.org/abstract?id=oai:eprints.rclis.org:2216.

D. Lewandowski, H. Wahlig, and G. Meyer-Bautor. The Freshness of Web Search Engine Databases. *Journal of Information Science*, 32(2):131–148, 2006. ISSN 0165-5515. doi: http://dx.doi.org/10.1177/0165551506062326.

L. Lim, M. Wang, S. Padmanabhan, J. S. Vitter, and R. C. Agarwal. Characterizing Web Document Change. In X. S. Wang, G. Yu, and H. Lu, editors, *WAIM '01: Proceedings of the Second International Conference on Advances in Web-Age Information Management*, volume 2118 of *Lecture Notes in Computer Science*, pages 133–144, Xi'an, China, July 9–11 2001. Springer-Verlag, London, United Kingdom. ISBN 3-540-42298-6. URL http://link.springer.de/link/service/series/0558/bibs/2118/21180133.htm.

K.-I. Lin and H. Chen. Automatic Information Discovery from the "Invisible Web". In *2002 International Symposium on Information Technology: Coding and Computing (ITCC 2002)*, pages 332–337, Las Vegas, Nevada, USA, Apr. 8–10 2002. IEEE Computer Society, Washington DC, Washington, USA. ISBN 0-7695-1506-1. URL http://csdl.computer.org/comp/proceedings/itcc/2002/1506/00/15060332abs.htm.

L. Liu, C. Pu, and W. Tang. WebCQ: Detecting and Delivering Information Changes on the Web. In *CIKM '00: Proceedings of the Ninth International Conference on Information and Knowledge Management*, pages 512–519, McLean, Virginia, USA, 2000. ACM Press, New York, New York, USA. ISBN 1-58113-320-0. doi: http://doi.acm.org/10.1145/354756.354860.

H. P. Luhn. A Statistical Approach to Mechanized Encoding and Searching of Literary Information. *IBM Journal of Research and Development*, 1 (4):309–317, 1957. URL http://domino.research.ibm.com/tchjr/journalindex.nsf/0/ff069b3bf042cc5e85256bfa00683d19?OpenDocument.

Lycos, Mar. 2008. URL http://www.lycos.com/.

G. S. Machovec. Veronica: A Gopher Navigational Tool on the Internet. *Information Intelligence, Online Libraries, and Microcomputers*, 11(10):1–4, Oct. 1 1993. ISSN 0737-7770.

C. Malcolm and G. Armitage. Bandwidth Efficient Web Object Change Interval Estimation. In *Australian Telecommunications Networks & Applications Conference 2003 (ATNAC 2003)*, Melbourne, Australia, Dec. 2003. URL http://citeseer.ist.psu.edu/666170.html.

J. Markwell and D. W. Brooks. Link Rot Limits the Usefulness of Web-Based Educational Material in Biochemistry and Molecular Biology. *Biochemistry and Molecular Biology Education*, 31:69–72, 2003.

O. A. McBryan. GENVL and WWWW: Tools for Taming the Web. In O. Nierstrasz, editor, *Proceedings of the First International World Wide Web Conference*, Geneva, Switzerland, May 25–27 1994.

A. K. McCallum, K. Nigam, J. Rennie, and K. Seymore. Automating the Construction of Internet Portals with Machine Learning. *Information Retrieval*, 3(2):127–163, 2000. ISSN 1386-4564. doi: http://dx.doi.org/10.1023/A:1009953814988.

F. McCown, X. Liu, M. L. Nelson, and M. Zubair. Search Engine Coverage of the OAI-PMH Corpus. *IEEE Internet Computing*, 10(2):66–73, 2006. ISSN 1089-7801. doi: http://dx.doi.org/10.1109/MIC.2006.41.

J. McDonnell, W. Koehler, and B. Carroll. Cataloging Challenges in an Area Studies Virtual Library Catalog (ASVLC): Results of a Case Study. *Journal of Internet Cataloging*, 2(2):15–42, 2000.

G. L. McLearn. Autonomous Cooperating Web Crawlers. Master's thesis, University of Waterloo, 2002. URL http://citeseer.nj.nec.com/mclearn02autonomous.html.

F. Menczer, G. Pant, P. Srinivasan, and M. E. Ruiz. Evaluating Topic-Driven Web Crawlers. In *SIGIR '01: Proceedings of the Twenty-Fourth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 241–249, New Orleans, Louisiana, USA, Sept. 9–13 2001. ACM Press, New York, New York, USA. ISBN 1-58113-331-6. doi: http://doi.acm.org/10.1145/383952.383995.

F. Menczer, G. Pant, and P. Srinivasan. Topical Web Crawlers: Evaluating Adaptive Algorithms. *ACM Transactions on Internet Technology (TOIT)*, 4(4):378–419, 2004. ISSN 1533-5399. doi: http://doi.acm.org/10.1145/1031114.1031117.

M. Mikhailov and C. Wills. Exploiting Object Relationships for Deterministic Web Object Management. In *Proceedings of the Seventh International Workshop on Web Content Caching and Distribution*, Boulder, Colorado, USA, Aug. 14–16 2002. URL http://citeseer.ist.psu.edu/mikhailov02exploiting.html.

M. Mikhailov and C. E. Wills. Evaluating a New Approach to Strong Web Cache Consistency with Snapshots of Collected Content. In *WWW '03: Proceedings of the Twelfth International Conference on World Wide Web*, pages 599–608, Budapest, Hungary, May 20–24 2003. ACM Press, New York, New York, USA. ISBN 1-58113-680-3. doi: http://doi.acm.org/10.1145/775152.775237.

R. C. Miller and K. Bharat. SPHINX: A Framework for Creating Personal, Site-Specific Web Crawlers. In P. H. Enslow Jr. and A. Ellis, editors, *WWW7: Proceedings of the Seventh International Conference on World Wide Web*, pages 119–130, Brisbane, Australia, Apr. 14–18 1998. Elsevier Science Publishers B. V., Amsterdam, The Netherlands. doi: http://dx.doi.org/10.1016/S0169-7552(98)00064-6.

A. Moffat, W. Webber, J. Zobel, and R. A. Baeza-Yates. A Pipelined Architecture for Distributed Text Query Evaluation. *Information Retrieval*, 10(3):205–231, June 2006. ISSN 1386-4564. doi: http://dx.doi.org/10.1007/s10791-006-9014-4.

J. Mogul. Errors in Timestamp-Based HTTP Header Values. Technical Report 99/3, Compaq Computer Corporation Western Research Laboratory, Dec. 1999. URL http://citeseer.ist.psu.edu/mogul99errors.html.

MSN, Mar. 2008. URL http://www.msn.com/.

MySQL, Sept. 2006. URL http://www.mysql.com/.

M. Najork and J. L. Wiener. Breadth-First Crawling Yields High-Quality Pages. In *WWW '01: Proceedings of the Tenth International World Wide Web Conference*, pages 114–118,

Hong Kong, China, May 1–5 2001. ACM Press, New York, New York, USA. ISBN 1-58113-348-0. URL http://citeseer.ist.psu.edu/najork01breadthfirst.html.

Netcraft. Netcraft Web Server Survey, 2006. URL http://www.netcraft.com/survey/.

Northern Light, Mar. 2008. URL http://www.northernlight.com/.

G. Notess. Search Engine Statistics: Freshness Showdown, May 17 2003. URL http://www.searchengineshowdown.com/statistics/freshness.shtml.

A. Ntoulas, J. Cho, and C. Olston. What's New on the Web? The Evolution of the Web from a Search Engine Perspective. In S. I. Feldman, M. Uretsky, M. Najork, and C. E. Wills, editors, *WWW '04: Proceedings of the Thirteenth International Conference on World Wide Web*, pages 1–12, New York, New York, USA, May 17–20 2004a. ACM Press, New York, New York, USA. ISBN 1-58113-844-X. doi: http://doi.acm.org/10.1145/988672.988674.

A. Ntoulas, P. Zerfos, and J. Cho. Downloading Hidden Web Content. Technical report, UCLA, 2004b.

A. Ntoulas, P. Zerfos, and J. Cho. Downloading Textual Hidden Web Content through Keyword Queries. In M. Marlino, T. Sumner, and F. M. S. III, editors, *ACM/IEEE Joint Conference on Digital Libraries, JCDL 2005*, pages 100–109, Denver, California, USA, June 7–11 2005. ACM Press. ISBN 1-58113-876-8. doi: http://doi.acm.org/10.1145/1065385.1065407.

C. Olston and J. Widom. Offering a Precision-Performance Tradeoff for Aggregation Queries over Replicated Data. In A. E. Abbadi, M. L. Brodie, S. Chakravarthy, U. Dayal, N. Kamel, G. Schlageter, and K.-Y. Whang, editors, *VLDB 2000, Proceedings of Twenty Sixth International Conference on Very Large Data Bases*, pages 144–155, Cairo, Egypt, Sept. 10–14 2000. Morgan Kaufmann. ISBN 1-55860-715-3.

C. Olston and J. Widom. Best-Effort Cache Synchronization with Source Cooperation. In M. J. Franklin, B. Moon, and A. Ailamaki, editors, *SIGMOD '02: Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, pages 73–84, Madison, Wisconsin, USA, June 3–6 2002. ACM Press, New York, New York, USA. ISBN 1-58113-497-5. doi: http://doi.acm.org/10.1145/564691.564701.

V. N. Padmanabhan and L. Qiu. The Content and Access Dynamics of a Busy Web Site: Findings and Implications. In *SIGCOMM '00: Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 111–123, Stockholm, Sweden, 2000. ACM Press, New York, New York, USA. ISBN 1-58113-223-9. doi: http://doi.acm.org/10.1145/347059.347413.

L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical report, Stanford Digital Library Technologies Project, 1998. URL http://citeseer.ist.psu.edu/page98pagerank.html.

S. Pandey and C. Olston. User-Centric Web Crawling. In A. Ellis and T. Hagino, editors, *WWW '05: Proceedings of the Fourteenth International Conference on World Wide Web*, pages 401–411, Chiba, Japan, May 10–14 2005. ACM Press, New York, New York, USA. ISBN 1-59593-046-9. doi: http://doi.acm.org/10.1145/1060745.1060805.

B. Pinkerton. Finding What People Want: Experiences with the WebCrawler. In *Proceedings of the Second International World Wide Web Conference*, Chicago, Illinois, USA, Oct. 1994.

P. Pirolli, J. Pitkow, and R. Rao. Silk from a Sows Ear: Extracting Usable Structures from the Web. In M. J. Tauber, editor, *CHI '96: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 118–125, Vancouver, Canada, Apr. 13–18 1996. ACM Press, New York, New York, USA. ISBN 0-89791-777-4. doi: http://doi.acm.org/10.1145/238386.238450.

J. E. Pitkow and P. Pirolli. Life, Death, and Lawfulness on the Electronic Frontier. In S. Pemberton, editor, *CHI '97: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 383–390, Atlanta, Georgia, USA, Mar. 22–27 1997. ACM Press/Addison-Wesley, New York, New York, USA. ISBN 0-201-32229-3. doi: http://doi.acm.org/10.1145/258549.258805.

V. Plachouras, I. Ounis, and G. Amati. A Utility-Oriented Hyperlink Analysis Model for the Web. In *LA-WEB '03: Proceedings of the First Conference on Latin American Web Congress, Empowering Our Web*, pages 123–131, Santiago, Chile, Nov. 10–12 2003. IEEE Computer Society, Washington DC, Washington, USA. ISBN 0-7695-2058-8.

S. Podlipnig and L. Böszörményi. A Survey of Web Cache Replacement Strategies. *ACM Computing Surveys*, 35(4):374–398, 2003. ISSN 0360-0300. doi: http://doi.acm.org/10.1145/954339.954341.

J. M. Ponte and W. B. Croft. A Language Modeling Approach to Information Retrieval. In *SIGIR '98: Proceedings of the Twenty First Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 275–281, Melbourne, Australia, 1998. ACM Press, New York, New York, USA. ISBN 1-58113-015-5. doi: http://doi.acm.org/10.1145/290941.291008.

M. F. Porter. An Algorithm for Suffix Stripping. *Readings in Information Retrieval*, pages 313–316, 1997.

M. Preda, S. Abiteboul, and G. Cobena. Adaptive On-Line Page Importance Computation. In *WWW '03: Proceedings of the Twelfth International Conference on World Wide Web*, pages 280–290, Budapest, Hungary, May 20–24 2003. ACM Press, New York, New York, USA. ISBN 1-58113-680-3. doi: http://doi.acm.org/10.1145/775152.775192.

C. Pu and A. Leff. Replica Control in Distributed Systems: An Asynchronous Approach. In J. Clifford and R. King, editors, *Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data*, pages 377–386, Denver, Colorado, USA, May 29–31 1991. ACM Press. doi: http://doi.acm.org/10.1145/115790.115856.

S. Raghavan and H. García-Molina. Crawling the Hidden Web. In *VLDB '01: Proceedings of the Twenty Seventh International Conference on Very Large Data Bases*, pages 129–138. Morgan Kaufmann Publishers Inc., San Francisco, California, USA, 2001. ISBN 1-55860-804-4.

J. Rennie and A. McCallum. Using Reinforcement Learning to Spider the Web Efficiently. In I. Bratko and S. Dzeroski, editors, *ICML '99: Proceedings of the Sixteenth International Conference on Machine Learning*, pages 335–343, Bled, Slovenia, June 27–30 1999. Morgan Kaufmann Publishers Inc., San Francisco, California, USA. ISBN 1-55860-612-2.

K. M. Risvik and R. Michelsen. Search Engines and Web Dynamics. *Computer Networks*, 39:289–302, June 2002. URL http://citeseer.ist.psu.edu/risvik02search.html.

R. Rivest. The MD5 Message-Digest Algorithm. Technical Report Internet RFC-1321, IETF, Apr. 1992. URL http://www.ietf.org/rfc/rfc1321.txt.

S. E. Robertson and K. S. Jones. Relevance Weighting of Search Terms. *Journal of the American Society for Information Science*, 27:129–146, 1976.

S. E. Robertson and S. Walker. Okapi/Keenbow at TREC-8. In *The Eighth Text REtrieval Conference (TREC-8)*, pages 151–162, Gaithersburg, Maryland, USA, Nov. 17–19 1999. National Institute of Standards and Technology (NIST), Washington DC, Washington, USA. URL http://trec.nist.gov/pubs/trec8/papers/okapi.pdf.

S. E. Robertson, S. Walker, S. Jones, M. Hancock-Beaulieu, and M. Gatford. Okapi at TREC-2. In D. Harman, editor, *The Second Text REtrieval Conference (TREC-2)*, pages 21–34, Gaithersburg, Maryland, USA, 1993. National Institute of Standards and Technology (NIST), Washington DC, Washington, USA. URL http://trec.nist.gov/pubs/trec2/papers/ps/city.ps.

G. Salton. *The SMART Retrieval System – Experiments in Automatic Document Processing*. Prentice Hall, Englewood Cliffs, New Jersey, USA, 1971.

G. Salton and C. Buckley. Term-Weighting Approaches in Automatic Text Retrieval. *Information Processing and Management*, 24(5):513–523, 1988. doi: http://dx.doi.org/10.1016/0306-4573(88)90021-0.

G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill Inc., New York, New York, USA, 1986. ISBN 0070544840.

A. Schaale, C. Wulf-Mathies, and S. Lieberam-Schmidt. A New Approach to Relevancy in Internet Searching - The "Vox Populi Algorithm". *ArXiv Computer Science e-prints*, Aug. 2003. URL http://adsabs.harvard.edu/cgi-bin/nph-bib_query?bibcode=2003cs.....8039S&db_key=PRE.

SEO Logic. Search Engine Optimization: Search Engine Marketing FAQ – Hidden Text, Sept. 2006. URL http://seologic.com/faq/hidden-text.php.

SEO Today. Google: Search Technology for the Millennium, 2002. URL http://www.seotoday.com/browse.php/category/articles/id/172/index.php.

N. Shivakumar and H. García-Molina. SCAM: A Copy Detection Mechanism for Digital Documents. In *Proceedings of the Second International Conference in Theory and Practice of Digital Libraries (DL '95)*, Austin, Texas, USA, June 1995.

V. Shkapenyuk and T. Suel. Design and Implementation of a High-Performance Distributed Web Crawler. In *Proceedings of the Eighteenth International Conference on Data Engineering (ICDE'02)*, pages 357–368. IEEE Computer Society, Washington DC, Washington, USA, 2002. URL http://citeseer.ist.psu.edu/shkapenyuk02design.html.

M. Shokouhi and P. Chubak. Designing a Regional Crawler for Distributed and Centralized Search Engines. In *Proceedings of Tenth Australian World Wide Web Conference (Ausweb 04)*, Gold Coast, Australia, July 3–7 2004.

S. Sivasubramanian, M. Szymaniak, G. Pierre, and M. van Steen. Replication for Web Hosting Systems. *ACM Computing Surveys*, 36(3):291–334, 2004. ISSN 0360-0300. doi: http://doi.acm.org/10.1145/1035570.1035573.

W. Sonnenreich and T. Macinta. *Web Developer.com ® Guide to Search Engines*. John Wiley & Sons, New York, New York, USA, 1998.

C. Spearman. The Proof and Measurement of Association Between Two Rings. *American Journal of Psychology*, 15:72–101, 1904.

D. Spinellis. The Decay and Failures of Web References. *Communications of the ACM (CACM)*, 46(1):71–77, 2003. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/602421.602422.

A. Spink, D. Wolfram, M. B. J. Jansen, and T. Saracevic. Searching the Web: The Public and their Queries. *Journal of the American Society for Information Science*, 52(3):226–234, 2001. ISSN 1532-2882. doi: http://dx.doi.org/10.1002/1097-4571(2000)9999:9999⟨::AID-ASI1591⟩3.3.CO;2-I.

P. Srinivasan, G. Pant, and F. Menczer. A General Evaluation Framework for Topical Crawlers. *Information Retrieval*, 8(3):417–447, 2005. ISSN 1386-4564. doi: http://dx.doi.org/10.1007/s10791-005-6993-5.

R. Srinivasan, C. Liang, and K. Ramamritham. Maintaining Temporal Coherency of Virtual Data Warehouses. In *RTSS '98: Proceedings of the IEEE Real-Time Systems Symposium*, page 60. IEEE Computer Society, Washington DC, Washington, USA, 1998. ISBN 0-8186-9212-X.

D. Sullivan. Search Engine Watch: Death of a Meta Tag, Oct. 1 2002. URL http://searchenginewatch.com/sereport/article.php/2165061.

D. Sullivan. Search Engine Watch, Sept. 2006a. URL http://searchenginewatch.com/.

D. Sullivan. Search Engine Watch: How Search Engines Rank Web Pages, July 31 2003a. URL http://searchenginewatch.com/showPage.html?page=2167961.

D. Sullivan. Search Engine Watch: Searches Per Day, Apr. 20 2006b. URL http://searchenginewatch.com/reports/article.php/2156461.

D. Sullivan. Search Engine Watch: Where are they now? Search Engines we've Known & Loved, Mar. 4 2003b. URL http://searchenginewatch.com/sereport/article.php/2175241.

R. Sundaresan, T. M. Kurç, M. Lauria, S. Parthasarathy, and J. H. Saltz. A Slacker Coherence Protocol for Pull-based Monitoring of On-line Data Sources. In *CCGRID '03: Proceedings of the Third International Symposium on Cluster Computing and the Grid*, pages 250–257, Tokyo, Japan, May 12–15 2003. IEEE Computer Society, Washington DC, Washington, USA. ISBN 0-7695-1919-9.

H. M. Taylor and S. Karlin. *An Introduction to Stochastic Modeling.* Academic Press, Orlando, Florida, USA, third edition, 1998.

Teoma, Sept. 2004. URL http://www.teoma.com/.

A. Tombros and M. Sanderson. Advantages of Query Biased Summaries in Information Retrieval. In *SIGIR '98: Proceedings of the Twenty-First Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 104–111, Melbourne, Australia, 1998. ACM Press, New York, New York, USA. ISBN 1-58113-015-5. doi: http://doi.acm.org/10.1145/290941.290947.

J. A. Tomlin. A New Paradigm for Ranking Pages on the World Wide Web. In *WWW '03: Proceedings of the Twelfth International Conference on World Wide Web*, pages 350–355, Budapest, Hungary, May 20–24 2003. ACM Press, New York, New York, USA. ISBN 1-58113-680-3. doi: http://doi.acm.org/10.1145/775152.775202.

T. Upstill, N. Craswell, and D. Hawking. Predicting Fame and Fortune: PageRank or Indegree? In *Proceedings of the Australasian Document Computing Symposium, ADCS2003*, pages 31–40, Canberra, Australia, December 2003a. URL http://research.microsoft.com/users/nickcr/pubs/upstill_adcs03.pdf.

T. Upstill, N. Craswell, and D. Hawking. Query-Independent Evidence in Home Page Finding. *ACM Transactions on Information Systems (TOIS)*, 21(3):286–313, 2003b. ISSN 1046-8188. doi: http://doi.acm.org/10.1145/858476.858479.

E. M. Voorhees and D. Harman, editors. *TREC: Experiment and Evaluation in Information Retrieval*. Digital Libraries and Electronic Publishing Series. MIT press, Cambridge, Massachusetts, USA, 2005. URL http://mitpress.mit.edu/catalog/item/default.asp?ttype=2&tid=10667.

W3C. W3C httpd: Proxies, July 1995. URL http://www.w3.org/Daemon/User/Config/Caching.html#CacheLastModifiedFactor.

W3C. The HTTP Distribution and Replication Protocol (DRP) Submission, Submitted by Marimba, Inc., Netscape Communications Corp., Novell, Inc., Sun Microsystems, Aug. 1997. URL http://www.w3.org/TR/NOTE-drp.

W3C. HTML 4.01 Specification, Dec. 1999. URL http://www.w3.org/TR/html40/.

D. Wessels. Squid Frequently Asked Questions: Caching, 2001a. URL http://www.squid-cache.org/Doc/FAQ/FAQ.pdf.

D. Wessels. Squid Internet Object Cache: Caching, 2001b. URL http://www.squid-cache.org/.

H. E. Williams and J. Zobel. Searchable Words on the Web. *International Journal of Digital Libraries*, 5(2):99–105, 2005. doi: http://dx.doi.org/10.1007/s00799-003-0050-z.

C. E. Wills and M. Mikhailov. Towards a Better Understanding of Web Resources and Server Responses for Improved Caching. *Computer Networks*, 31(11–16):1231–1243, 1999a. ISSN 1389-1286. doi: http://dx.doi.org/10.1016/S1389-1286(99)00037-7.

C. E. Wills and M. Mikhailov. Examining the Cacheability of User-Requested Web Resources. In *Proceedings of the Fourth International Web Caching Workshop*, San Diego, California, USA, Apr. 1999b. URL http://citeseer.ist.psu.edu/article/wills99examining.html.

I. H. Witten, A. Moffat, and T. C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann Publishing, San Francisco, California, USA, second edition, May 1999. ISBN 1-55860-570-3.

J. L. Wolf, M. S. Squillante, P. S. Yu, J. Sethuraman, and L. Ozsen. Optimal Crawling Strategies for Web Search Engines. In *WWW '02: Proceedings of the Eleventh International Conference on World Wide Web*, pages 136–147, Honolulu, Hawaii, USA, May 7–11 2002. ACM Press, New York, New York, USA. ISBN 1-58113-449-5. doi: http://doi.acm.org/10.1145/511446.511465.

W. Wu, C. Yu, A. Doan, and W. Meng. An Interactive Clustering-Based Approach to Integrating Source Query Interfaces on the Deep Web. In G. Weikum, A. C. König, and S. Deßloch, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 95–106, Paris, France, June 13–18 2004. ACM Press, New York, New York, USA. ISBN 1-58113-859-8. doi: http://doi.acm.org/10.1145/1007568.1007582.

Yahoo!, Mar. 2008. URL http://www.yahoo.com/.

H. Yu and A. Vahdat. Efficient Numerical Error Bounding for Replicated Network Services. In A. E. Abbadi, M. L. Brodie, S. Chakravarthy, U. Dayal, N. Kamel, G. Schlageter, and K.-Y. Whang, editors, *VLDB 2000, Proceedings of Twenty Sixth International Conference on Very Large Data Bases*, pages 144–155, Cairo, Egypt, Sept. 10–14 2000. Morgan Kaufmann. ISBN 1-55860-715-3.

S. Zander, G. Armitage, and C. Malcolm. Dynamics and Cachability of Web Sites: Implications for Inverted Capacity Networks. In B. Landfeldt and T. Moors, editors, *The*

Eleventh IEEE International Conference on Networks (ICON2003), pages 45–50, Sydney, Australia, Sept. 28– Oct. 1 2003. IEEE Computer Society Press. ISBN 0-7803-7788-5.

D. Zeinalipour-Yazti and M. D. Dikaiakos. Design and Implementation of a Distributed Crawler and Filtering Processor. In A. Y. Halevy and A. Gal, editors, *NGITS '02: Proceedings of the Fifth International Workshop on Next Generation Information Technologies and Systems*, volume 2382 of *Lecture Notes in Computer Science*, pages 58–74, Caesarea, Israel, June 24–25 2002. Springer-Verlag, London, United Kingdom. ISBN 3-540-43819-X. URL http://link.springer.de/link/service/series/0558/bibs/2382/23820058.htm.

Zettair, Mar. 2008. URL http://www.seg.rmit.edu.au/zettair/.

C. Zhai and J. Lafferty. A Study of Smoothing Methods for Language Models Applied to Information Retrieval. *ACM Transactions on Information Systems (TOIS)*, 22(2):179–214, 2004. ISSN 1046-8188. doi: http://doi.acm.org/10.1145/984321.984322.

Y. Zhang, P. Vines, and J. Zobel. Chinese OOV Translation and Post-Translation Query Expansion in Chinese–English Cross-Lingual Information Retrieval. *ACM Transactions on Asian Language Information Processing (TALIP)*, 4(2):57–77, 2005. ISSN 1530-0226. doi: http://doi.acm.org/10.1145/1105696.1105697.

J. Zobel and A. Moffat. Inverted Files for Text Search Engines. *ACM Computing Surveys*, 38(2):6, 2006. ISSN 0360-0300. doi: http://doi.acm.org/10.1145/1132956.1132959.