

UNIVERSITY OF CALIFORNIA
Los Angeles

Crawling and Searching the Hidden Web

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Computer Science

by

Alexandros Ntoulas

2006

© Copyright by
Alexandros Ntoulas
2006

The dissertation of Alexandros Ntoulas is approved.

Gregory H. Leazer

D. Stott Parker, Jr.

Carlo Zaniolo

Junghoo Cho, Committee Chair

University of California, Los Angeles

2006

Στους Γονείς μου

TABLE OF CONTENTS

1	Introduction	1
1.1	Challenges in creating a Hidden-Web search engine	3
1.2	Organization and contributions of the dissertation	5
2	Crawling the Hidden Web	10
2.1	Introduction	10
2.2	Framework	11
2.2.1	Hidden-Web database model	11
2.2.2	A generic Hidden-Web crawling algorithm	13
2.2.3	Problem formalization	15
2.3	Keyword Selection	18
2.3.1	Estimating the number of matching pages	19
2.3.2	Query selection algorithm	22
2.3.3	Efficient calculation of query statistics	25
2.3.4	Crawling sites that limit the number of results	27
2.4	Experimental Evaluation	29
2.4.1	Comparison of policies	32
2.4.2	Impact of the initial query	38
2.4.3	Impact of the limit in the number of results	40
2.4.4	Incorporating the document download cost	42
2.4.5	Other observations	43

2.5	Related Work	45
2.5.1	Cooperative data sources	47
2.5.2	Uncooperative data sources	49
2.6	Summary	51
3	Evolution of Searchable Web Sites	53
3.1	Introduction	53
3.2	Experimental setup	54
3.2.1	Selection of the sites	55
3.2.2	Download of pages	55
3.3	What's new on the Web?	56
3.3.1	Weekly birth rate of pages	57
3.3.2	Birth, death, and replacement	58
3.3.3	The creation of new content	66
3.3.4	Link-structure evolution	70
3.4	Changes in the existing pages	72
3.4.1	Change frequency distribution	73
3.4.2	Degree of change	74
3.4.3	Degree and frequency of change	80
3.5	Predictability of degree of change	83
3.5.1	Overall predictability	84
3.5.2	Predictability for individual sites	86
3.6	Related Work	87

3.7	Summary	90
4	Sampling-based Page Update	92
4.1	Introduction	92
4.2	Framework	93
4.2.1	Download policies	94
4.2.2	Evaluation metrics	96
4.3	Sampling-based policies	100
4.3.1	Sampling cost model	100
4.3.2	Greedy and proportional policies	100
4.3.3	Optimal sample size	102
4.3.4	Adaptive sampling	107
4.3.5	Subset sampling under low download resources	109
4.3.6	Is <i>Greedy</i> too greedy?	112
4.4	Experiments	113
4.4.1	Distribution of ρ values	114
4.4.2	Rough comparison of download policies	115
4.4.3	Optimal sample size	117
4.4.4	Resource size and subset sampling	118
4.4.5	Long-term performance of the frequency policy	120
4.4.6	Adaptive policy	121
4.4.7	How greedy is <i>Greedy</i> ?	123
4.4.8	Performance under different distributions	125

4.5	Related work	126
4.6	Summary	128
4.7	Appendix: Proofs of the theorems	129
5	Inverted Index Pruning with Correctness Guarantee	135
5.1	Introduction	135
5.2	Background	137
5.2.1	Overview of indexing and querying	137
5.2.2	Ranking of the relevant documents	139
5.3	Cluster architecture and cost savings from a pruned index	142
5.4	Optimal size of the p-index	147
5.5	Pruning policies	151
5.5.1	Horizontal pruning	152
5.5.2	Vertical pruning	155
5.6	Index maintenance	163
5.7	Experimental Evaluation	165
5.7.1	Data and experimental setup	165
5.7.2	Horizontal pruning	166
5.7.3	Vertical pruning	170
5.7.4	Optimal p-index size	174
5.7.5	Combining horizontal and vertical pruning	176
5.8	Related Work	179
5.9	Summary	181

6	Content-based Spam Detection	182
6.1	Introduction	182
6.2	Experimental framework and data set	185
6.3	How much spam?	187
6.4	Content-based spam detection	190
6.4.1	Number of words in the page	191
6.4.2	Number of words in the page title	192
6.4.3	Average length of words	194
6.4.4	Amount of anchor text	195
6.4.5	Fraction of visible content	197
6.4.6	Compressibility	200
6.4.7	Fraction of page drawn from globally popular words	202
6.4.8	Fraction of globally popular words	202
6.4.9	Independent n-gram likelihoods	204
6.4.10	Conditional n-gram likelihoods	207
6.5	Using classifiers to combine heuristics	210
6.6	Related Work	216
6.7	Summary	218
7	The <i>HiddenSeek</i> Hidden-Web Search Engine	219
7.1	Introduction	219
7.2	The Architecture of HiddenSeek	219
7.2.1	Crawling the Hidden-Web	221

7.2.2	The Link Database	221
7.2.3	Linguistic Processing	221
7.2.4	Inverted Indexes	222
7.2.5	Page Summaries	222
7.2.6	Answering a Query	223
7.2.7	User Interface	226
8	Conclusion and Future Work	227
8.1	Conclusion	227
8.2	Future Work	229
8.2.1	Hidden-Web Crawling	229
8.2.2	Update of Pages	229
8.2.3	Spam Detection	230
	References	232

LIST OF FIGURES

2.1	A single-attribute search interface	11
2.2	A multi-attribute search interface	12
2.3	Pages from the PubMed Web site.	12
2.4	Algorithm for crawling a Hidden-Web site.	14
2.5	A set-formalization of the optimal query selection problem.	15
2.6	Fitting a Zipf Distribution in order to estimate $P(q_i)$	23
2.7	Algorithm for selecting the next query term.	24
2.8	Updating the query statistics table.	26
2.9	A Web site that does not return all the results.	28
2.10	Coverage of policies for Pubmed	32
2.11	Coverage of policies for Amazon	33
2.12	Coverage of policies for general dmoz	34
2.13	Coverage of policies for the Arts section of dmoz	35
2.14	Convergence of the adaptive algorithm using different initial queries for crawling the PubMed Web site	39
2.15	Coverage of general dmoz after limiting the number of results to 1,000	41
2.16	Coverage of PubMed after incorporating the document download cost	44
3.1	Fraction of new pages between successive snapshots.	59
3.2	Fraction of pages from the first crawl still existing after n weeks (dark bars) and new pages (light bars).	60

3.3	Normalized fraction of pages from the first crawl still existing after n weeks (dark bars) and new pages (light bars).	61
3.4	Percent of IP addresses identifying a Web site in Year A also identifying a Web site in Year B. For example, 56% of IP addresses identifying a Web site in the 1998 sample also identified one in 1999 sample. Taken from http://wcp.oclc.org/	64
3.5	Average page sizes in our snapshots over time.	66
3.6	Fraction of shingles from the first crawl still existing after n weeks (dark portion of bars) and shingles newly created (light portion). . . .	68
3.7	Normalized fraction of shingles from the first crawl still existing after n weeks (dark portion of bars) and shingles newly created (light portion). . . .	69
3.8	Fraction of links from the first weekly snapshot still existing after n weeks (dark/bottom portion of the bars), new links from existing pages (grey/middle) and new links from new pages (white/top).	70
3.9	Normalized fraction of links from the first weekly snapshot still existing after n weeks (dark/bottom portion of the bars), new links from existing pages (grey/middle) and new links from new pages (white/top).	71
3.10	Distribution of the average change intervals of the pages.	74
3.11	Distribution of cosine distance for all changes. Each dark bar corresponds to changes with cosine distance between the respective x-axis value and the previous one. For example, bin 0.1 corresponds to changes with cosine distance between 0.05 and 0.1. The light bars show the cumulative distribution.	77

3.12	Distribution of word distance values for all changes. Each dark bar corresponds to changes with word distance between the respective x-axis value and the previous one. For example, bin 0.1 corresponds to changes with word distance between 0.05 and 0.1. The light bars show the cumulative distribution.	78
3.13	Relationship between degree of change and frequency of change. . . .	81
3.14	Relationship between <i>cumulative</i> degree of change and frequency of change.	82
3.15	(a) Cosine distance predictability over time for all pages, (b) only for the site <code>www.columbia.edu</code> , and (c) only for <code>www.eonline.com</code> . The first column represents changes falling one week apart, the second column shows changes one month apart, and the last column shows changes three months apart.	91
4.1	Expected <i>ChangeRatio</i> for various sample sizes	103
4.2	Histogram of ρ_i values of sites	105
4.3	Algorithm of the adaptive-sampling policy	108
4.4	Comparison of the round-robin policy and a sampling policy for various resource constraints	110
4.5	Histogram of ρ_i values in the dataset.	114
4.6	Comparison of the various download policies.	116
4.7	Performance of the greedy and proportional policies over various sample sizes.	118
4.8	Performance of oracle, greedy and round-robin policies over various resource sizes.	119

4.9	Comparison of <i>ChangeRatio</i> for <i>Greedy</i> and <i>SubsetSampling</i> for low number of resources.	120
4.10	Performance of the frequency and the greedy policies over time	120
4.11	Performance of <i>Adaptive</i> over various sample sizes k	122
4.12	Performance of <i>Adaptive</i> over various confidence values α	123
4.13	Fairness of <i>Greedy</i> policy.	124
4.14	Average number of refreshes over number of changes for <i>Greedy</i> policy.	125
4.15	Comparison of various policies under a normal distribution $N(0.5, 0.3)$ of ρ values.	127
4.16	Probability $P_o(\rho)$ and $P_g(\rho)$	130
5.1	(a) Search engine replicates its full index I_F to increase query-answering capacity. (b) In the 1^{st} tier, small p-indexes I_P handle most of the query. When I_P cannot answer a query, it is redirected to the 2^{nd} tier, where the full index I_F is used to compute the answer.	143
5.2	Computing the answer under the two-tier architecture with the result correctness guarantee.	146
5.3	Example function showing the fraction of guaranteed queries $f(s)$ at a given size s of the p-index.	150
5.4	Horizontal and vertical pruning. During horizontal pruning, we remove inverted lists from the index: inverted lists t_3 and t_5 are removed. During vertical pruning we remove postings from the inverted lists: postings D_4, D_5, D_6 from t_1 and posting D_8 from t_2 are removed. . .	153
5.5	Approximation algorithm for the optimal horizontal pruning.	154
5.6	Result guarantee in horizontal pruning.	155

5.7	Global vertical pruning based on pr	156
5.8	Local vertical pruning based on pr	157
5.9	Extended keyword-specific vertical pruning based on pr and tr	159
5.10	Ranking based on thresholds $tr_{\tau}(t_i)$ and $pr_{\tau}(t_i)$	160
5.11	Query-load stability over time.	164
5.12	Fraction of <i>guaranteed queries</i> $f(s)$ answered in a horizontally-pruned p-index of size s	167
5.13	Fraction of <i>guaranteed queries</i> $f(s)$ answered in a horizontally-pruned p-index of size $s = 0.3$ over time.	168
5.14	Fraction of <i>guaranteed queries</i> $f(s)$ answered in a vertically-pruned p-index of size s	169
5.15	Fraction of queries answered in a vertically-pruned p-index of size s . .	173
5.16	Average fraction of the top-20 results of p-index with size s contained in top-20 results of the full index.	175
5.17	Combining horizontal and vertical pruning.	176
5.18	Performance of combined horizontal and vertical pruning.	177
6.1	An example spam page; although it contains popular keywords, the overall content is useless to a human user.	186
6.2	Spam occurrence per top-level domain in our data set.	188
6.3	Spam occurrence per language in our data set.	189
6.4	Prevalence of spam relative to number of words on page.	193
6.5	Prevalence of spam relative to number of words in title on page. . . .	194
6.6	Prevalence of spam relative to average word length of page.	196

6.7	Prevalence of spam relative to amount of anchor text of page.	198
6.8	Prevalence of spam relative to fraction of visible content on page. . .	199
6.9	Prevalence of spam relative to compressibility of page.	201
6.10	Prevalence of spam relative to fraction of words on page that are among the most frequent 200 words in the entire corpus.	203
6.11	Prevalence of spam relative to fraction of the most frequent 500 words from corpus that are among the words of a page.	205
6.12	Prevalence of spam relative to independent likelihood of word 3-grams on the page.	208
6.13	Prevalence of spam relative to conditional likelihood of word 3-grams on the page.	209
6.14	A portion of the induced C4.5 decision tree for spam detection. The ellipsis indicate elided portions of the tree.	212
7.1	The architecture of HiddenSeek.	220
7.2	The homepage of <i>HiddenSeek</i> showing the default user interface . . .	224
7.3	After issuing query <code>thyroid cancer</code> to <i>HiddenSeek</i> the user sees a list of results from Hidden-Web sites. In this figure, results from <code>webmd.com</code> , <code>pubmed.com</code> and <code>uspto.gov</code> are shown.	225

LIST OF TABLES

2.1	Sample of keywords queried to PubMed exclusively by the adaptive policy	37
2.2	Related work organized according to whether the Hidden-Web sources are <i>cooperative</i> , and whether we are interested in information <i>integration</i> or <i>warehousing</i>	46
3.1	Distribution of domains in our crawls.	57
4.1	Notation used throughout Chapter 4.	105
5.1	Notation used throughout Chapter 5.	138
6.1	Recall and precision of our classifier.	213
6.2	Recall and precision after bagging.	215
6.3	Recall and precision after boosting.	215

ABSTRACT OF THE DISSERTATION

Crawling and Searching the Hidden Web

by

Alexandros Ntoulas

Doctor of Philosophy in Computer Science

University of California, Los Angeles, 2006

Professor Junghoo Cho, Chair

An ever-increasing amount of valuable information on the Web today is hidden behind search interfaces. This information is collectively called the Hidden Web.

In this dissertation, we study how we can effectively collect the data from the Hidden Web and enable the users to search for information within the collected data. More specifically, we address some of the main challenges involved in creating a search engine for the Hidden Web:

Crawling the Hidden Web. We study how to build an effective Hidden-Web crawler that can facilitate the collection of information from the Hidden Web. Since there are no links to the Hidden Web pages, our crawler needs to automatically come up with queries to issue to the Hidden Web sites. We propose three different query generation policies for the Hidden Web: a policy that picks queries at *random* from a list of keywords, a policy that picks queries based on their *frequency* in a generic text collection, and a policy which *adaptively* picks a good query based on the content of the pages downloaded from the Hidden-Web site. We compare the effectiveness of our policies by crawling a number of real Hidden-Web sites.

Updating the Hidden-Web pages. The information on the Web today is constantly evolving. Once our crawler has downloaded the information from the Hidden-

Web, it needs to periodically refresh its local copy in order to enable users to search for up-to-date information. We study the evolution of searchable Web sites using real data collected from the Web over a period of one year. We also propose an efficient sampling-based policy for updating the pages.

Indexing and Searching the Hidden Web. Once we have downloaded the Hidden-Web pages, we can enable the users to search for useful information. Search engines typically do this by maintaining large-scale *inverted indexes* which are replicated dozens of times for scalability and which are then pruned in order to reduce the cost of operation. We show that the current approaches employed by the search engines may result in significant degradation in the quality of results. To alleviate this problem, we propose modifications to current pruning techniques so that we avoid any degradation in quality, while realizing the benefit of lower cost of operation.

Fighting Web spam. In the last few years, several sites on the Web observe an ever-increasing portion of their traffic coming from search engine referrals. Given the large fraction of Web traffic originating from searches and the high potential monetary value of this traffic, some Web site operators try to influence the positioning of their pages within search results by crafting *spam* Web pages. In the case of the Hidden Web, these malicious Web site operators may try to pollute our index for their own benefit by injecting spam content in their Hidden-Web databases so that our crawler can download it. In this dissertation, we study the prevalence of spam on the Web and we present a number of techniques to detect Web spam. We also show how to use machine learning techniques to combine our techniques into creating a more effective spam detection mechanism.

The techniques proposed in this dissertation have been incorporated in a prototype search engine that currently indexes a few million pages from the Hidden Web.

CHAPTER 1

Introduction

An ever-increasing amount of information on the Web today cannot be reached by following links [Ber01, CHL03]. In particular, a large part of the Web is “hidden” behind search forms, and is reachable only when users type in a set of keywords, or *queries*, to the forms. These pages are often referred to as the *Hidden Web* [FLM98] or the *Deep Web* [Ber01], because search engines typically cannot index the pages and do not return them in their results (thus, the pages are essentially “hidden” from a typical Web user).

According to many studies, the size of the Hidden Web increases rapidly as more organizations make their valuable content accessible online through an easy-to-use Web interface [Ber01]. In [CHL03], Chang et al. estimate that well over 100,000 Hidden-Web sites currently exist on the Web. Moreover, the content provided by many Hidden-Web sites is often of very high quality and can be extremely valuable to many users [Ber01]. For example, PubMed [Pub] hosts many high-quality papers on medical research that were selected from careful peer-review processes, while the site of the US Patent and Trademarks Office [USP] makes existing patent documents available, helping potential inventors examine “prior art.”

In this dissertation, we study how we can build a search engine for the Hidden Web i.e., how we can effectively collect the data from the Hidden Web and enable the users to search for information within the collected data. A Hidden-Web search engine can bring tremendous benefits to the users including the following:

- **Tapping into unexplored information:** Our Hidden-Web search engine will allow an average Web user to easily explore the vast amount of information that is mostly “hidden” at present. Since a majority of Web users rely on search engines to discover pages, when pages are not indexed by search engines, they are unlikely to be viewed by many Web users. Unless users go directly to Hidden-Web sites and issue queries there, they cannot access the pages at the sites.
- **Improving user experience:** Even if a user is aware of a number of Hidden-Web sites, the user currently has to waste a significant amount of time and effort, visiting all of the potentially relevant sites, querying each of them and exploring the result. By making the Hidden-Web pages searchable at a central location, we can significantly reduce the user’s wasted time and effort in searching the Hidden Web.
- **Reducing potential bias:** Due to the heavy reliance of many Web users on search engines for locating information, search engines influence how the users perceive the Web [Ols02]. Users do *not* necessarily perceive what actually *exists* on the Web, but what is *indexed* by search engines [Ols02]. According to a recent article [New04], several organizations have recognized the importance of bringing information of their Hidden-Web sites onto the surface, and committed considerable resources towards this effort. By downloading the Hidden-Web information and making it accessible we hope to remove some of this bias.

1.1 Challenges in creating a Hidden-Web search engine

Given the enormous size of the Hidden Web and the fact that the information is only accessible through a search form, building a search engine for the Hidden Web imposes many important challenges, including the following:

1. **Crawling the Hidden Web.** *Crawlers* are programs that traverse the Web automatically and download pages for search engines. Conventional crawlers today rely mainly on the hyper-links on the Web to discover and download pages. Due to the lack of links pointing to the Hidden-Web pages, the current search engines cannot index the Hidden-Web pages.

Given that the only “entry” to the Hidden-Web pages is through querying a search form, the main challenge to implementing an effective Hidden-Web crawler is to automatically generate *meaningful* queries so that it can discover and download the Hidden-Web pages.

2. **Updating the Hidden-Web pages.** The information on the Web today is highly volatile. New and improved content is constantly added to serve the users’ needs in a better way. At the same time, the potentially useful content is removed from the Web at an administrator’s whim.

Once our crawler has downloaded the information from the Hidden-Web, it needs to periodically refresh its local copy in order to enable users to search for up-to-date information. However, because pages change independently and at different rates, it is very important for the crawler to select carefully which pages to refresh: if the crawler downloads pages that have not changed, then it is simply wasting time and bandwidth instead of updating our local copy. Therefore, one challenge that the crawler has to face is being able to identify which pages on the Hidden-Web have changed and download them in a timely manner.

3. **Indexing and searching the Hidden Web.** Once we have downloaded the Hidden-Web pages, we can enable the users to search for useful information. Search engines typically do this by maintaining large-scale *inverted indexes* which are replicated dozens of times for scalability. Given the enormous size of available information on the Hidden Web, this task can become very costly. One way to reduce the cost is to replace some of the full inverted indexes with smaller, *pruned* inverted indexes [CCF01, LS03], where the most frequently-accessed portions of the index are stored. While this approach may give significant improvement in performance, it also leads to noticeable degradation in the quality of the search results, when the top answers are computed only from the pruned index [CCF01, LS03].

Since our goal is to provide the users with Hidden-Web results of the highest quality, this degradation is clearly undesirable. Therefore, the challenge that we have to address is how to avoid *any* degradation of result quality due to the pruning-based performance optimization, while still realizing most of its benefit.

4. **Fighting Web spam.** In the last few years, several sites on the Web observe an ever-increasing portion of their traffic coming from search engine referrals. For many commercial Web sites, an increase in search engine referrals translates to an increase in sales, revenue, and profits. Given the large fraction of Web traffic originating from searches and the high potential monetary value of this traffic, some Web site operators try to influence the positioning of their pages within search results by crafting “bogus” or *spam* Web pages which aim solely at manipulating search engines and are useless to any human user.

In the case of the Hidden-Web, these malicious Web site operators may try to pollute our index by injecting spam content in their Hidden-Web databases so that our crawler can download it. Therefore, one challenge that we have to

address is to create an effective Web spam detection method, in order to ensure the quality of the data that we return to our users.

1.2 Organization and contributions of the dissertation

In this dissertation we address the aforementioned challenges by designing, implementing and evaluating the architectural components of a Hidden-Web search engine. As part of this dissertation we built a Hidden-Web search engine prototype called *HiddenSeek* which currently indexes a few million Hidden-Web pages from various sources online. In this dissertation we address the challenges that we encountered during the realization of *HiddenSeek* and we present our experimental and algorithmic solutions that address these challenges.

The remainder of this dissertation is organized as follows:

Chapter 2: Crawling the Hidden Web. We start by studying how we can build a Hidden-Web crawler that can automatically download pages from the Hidden Web, so that search engines can index them. In particular, we study how our crawler can automatically come up with meaningful queries for a search form in order to discover and download the Hidden-Web pages.

In Chapter 2, we provide a theoretical framework to investigate the Hidden-Web crawling problem and propose effective ways of generating queries automatically. We also evaluate our proposed solutions through experiments conducted on *real* Hidden-Web sites. In summary, in Chapter 2 we make the following contributions:

1. We present a formal framework to study the problem of Hidden-Web crawling.
2. We investigate a number of crawling policies for the Hidden Web, including the optimal policy that can potentially download the maximum number of pages

through the minimum number of interactions. We show that the optimal policy is NP-hard and cannot be implemented in practice.

3. We propose a new adaptive policy that approximates the optimal policy. Our adaptive policy examines the pages returned from previous queries and adapts its query-selection policy automatically based on them.
4. We evaluate various crawling policies through experiments on real Web sites. Our experiments show the relative advantages of various crawling policies and demonstrate their potential. The results from our experiments are very promising. In certain cases, our adaptive policy downloaded more than 90% of the pages within a Hidden-Web site after issuing fewer than 100 queries.

Chapter 3: Evolution of the Hidden Web. We study various aspects of the evolving Hidden Web, by monitoring pages in 154 Web sites that make their full set of pages available through search forms. We monitor the sites on a weekly basis for one year and we analyze the evolution of these sites.

Our contributions can be summarized as follows:

1. We estimate that new pages are created at the rate of 8% per week. Assuming that the current Web has at least 11.5 billion pages [GS05], this result corresponds to 920 million new pages every week.
2. We also estimate that only 20% of the pages available today will be still accessible after one year.
3. While a large number of new pages are created every week, the new pages seem to “borrow” a significant portion of their content from exiting pages. We observe that about 5% of “new content” is being introduced every week. Overall, after a year, about 50% of the content is new.

4. The link structure of the Web is significantly more dynamic than the content on the Web. Every week, about 25% new links are created. After a year, about 80% of the links on the Web are replaced with new ones.
5. Our results indicate that once a page is created, the page is likely to go through either minor changes or no change at all. Out of all pages that are still available after one year, half of them do not change at all during that year. Even for the pages that do change, the changes are very minor.
6. *Frequency of change*: We show that the frequency of change of the pages is not a good “predictor” of the degree of change in the pages. This result implies that existing refresh algorithms for search engines may not be a good choice if we want to maximize the degree of change that the search engines detect. Most existing algorithms use the frequency of change as their prediction mechanism [CG00a, CLW98].
7. *Degree of change*: The past degree of change of a page exhibits a strong correlation with the future degree of change. We also observe that the correlation varies significantly between the sites. While some sites exhibit a very strong correlation some sites do not.

Chapter 4: Sampling-based Page Update. In Chapter 4 we study how we can detect and download as many changed Web pages as we can, when the pages are updated independently. Our main idea to address this challenge is to use *sampling*. That is, we first download a small number of data items from each data source which serve as *samples*. After that, we use the samples to decide which of the sources we should contact in order to download more data items.

Our contributions are the following:

1. We present a framework to study the change detection and download problem. We discuss various change-detection policies and present evaluation metrics for comparing different policies.
2. We propose two sampling-based policies, *proportional* and *greedy*, and analyze their effectiveness. We also derive the optimal sample size that maximizes the effectiveness of a sampling-based policy.
3. We propose an adaptive-sampling policy that can dynamically adjust the sample size, based on the changes detected so far. We also investigate the scenario where we cannot sample enough pages from each data source due to our very limited download resources.
4. We experimentally compare our sampling-based policies to others, including the state-of-the-art frequency-based policy. Our experiments show that our sampling policy is often significantly better than existing ones.

Chapter 5: Inverted Index Pruning with Correctness Guarantee. In Chapter 5 we study how we can efficiently index and search the enormous amounts of information on the both the Visible and the Hidden Web while avoiding *any* degradation of search quality due to index pruning.

Overall, we make the following contributions:

1. We study new pruning techniques and answer computation algorithms that *guarantee* that the top matching pages are *always* placed at the top of search results, even if we compute the first batch of answers from a pruned index most of the time.
2. We provide a framework to determine the optimal size of the pruned index for top-answer computation that minimizes the overall cost of answering queries

under the cluster architecture.

3. We provide theoretical justification of our proposed techniques and experimental evaluation on real Web data.

Chapter 6: Content-based Spam Detection. In Chapter 6, we explore a variety of methods for detecting spam. Each method is highly parallelizable, can run in time proportional to the size of the page, and identifies spam pages by analyzing the content of every downloaded page.

Overall, we make the following contributions:

1. We present experiments performed on real pages collected from the Web that demonstrate the amount of spam on the Web, as well as the relative merits of every spam-detection method.
2. We demonstrate how to employ machine learning techniques that combine our individual methods to create a highly efficient and reasonably-accurate spam detection algorithm.
3. When combined, our methods correctly identify 86.2% of the spam pages while misidentifying only 3.1% of all the pages.

Chapter 7: The *HiddenSeek* Hidden-Web Search Engine. Finally, in Chapter 7 we conclude by describing the overall architecture of a Hidden-Web search engine prototype which incorporates the research contributions presented in this dissertation. To the best of our knowledge this is the first search engine that enables the users to search information from the Hidden Web.

CHAPTER 2

Crawling the Hidden Web

2.1 Introduction

In this chapter, we study how we can build an effective *Hidden-Web crawler* that can automatically download pages from the Hidden Web, so that a search engine can index them. *Crawlers* are the programs that traverse the Web automatically and download pages for search engines. Conventional crawlers rely on the hyperlinks on the Web to discover pages. Unfortunately, since there are no links pointing to the Hidden-Web pages, current search engines cannot index those pages and do not return them to the users.

Given that the only “entry” to Hidden-Web pages is through querying a search form, there are two core challenges to implementing an effective Hidden-Web crawler: (a) The crawler has to be able to understand and model a query interface, and (b) The crawler has to come up with meaningful queries to issue to the query interface. The first challenge was addressed by Raghavan and Garcia-Molina in [RG01], where a method for learning search interfaces was presented. Here, we present a solution to the second challenge, i.e. how a crawler can automatically generate queries so that it can discover and download the Hidden-Web pages.

Clearly, when the search forms list all possible values for a query (e.g., through a drop-down list), the solution is straightforward. We exhaustively issue all possible queries, one query at a time. When the query forms have a “free text” input, however,



Figure 2.1: A single-attribute search interface

an *infinite* number of queries are possible, so we cannot exhaustively issue all possible queries. In this case, what queries should we pick? How can the crawler automatically come up with meaningful queries without understanding the semantics of the search form?

In this chapter, we provide a theoretical framework to investigate the Hidden-Web crawling problem and propose effective ways of generating queries automatically. We also evaluate our proposed solutions through experiments conducted on *real* Hidden-Web sites.

2.2 Framework

We start by presenting a formal framework for the study of the Hidden-Web crawling problem. In Section 2.2.1, we describe our assumptions on Hidden-Web sites and explain how users interact with the sites. Based on this interaction model, we present a high-level algorithm for a Hidden-Web crawler in Section 2.2.2. Finally in Section 2.2.3, we formalize the Hidden-Web crawling problem.

2.2.1 Hidden-Web database model

There exists a variety of Hidden Web sources that provide information on a multitude of topics. Depending on the type of information, we may categorize a Hidden-Web site either as a *textual database* or a *structured database*. A textual database is a site that mainly contains plain-text documents, such as *PubMed* and *Lexis-Nexis* (an online database of legal documents [Lex]). Since plain-text documents do not usually have

Author:

Title:

ISBN:

Search Now

Figure 2.2: A multi-attribute search interface

Display

Summary

Show: 20

Sort

Send to

Text

Items 1-20 of 592376

Page 1 of 29619

Next

1

Cruz A, Padillo FJ, Granados J, Tunez I, Munoz MC, Briceno J, Pera-Madrado C, Montilla P.

Related Articles, Links

Effect of melatonin on cholestatic oxidative stress under constant light exposure.

Cell Biochem Funct. 2003 Dec;21(4):377-80.

PMID: 14624477 [PubMed - in process]

2

Erisir M, Beyrut E, Ozan S, Aksakal M.

Related Articles, Links

Effects of dietary vitamin E and selenium on arginase activity in the liver, kidneys, and heart of rats treated with high doses of glucocorticoid.

Cell Biochem Funct. 2003 Dec;21(4):331-5.

PMID: 14624471 [PubMed - in process]

3

Yilmaz S, Ozan S, Benzer F, Canatan H.

Related Articles, Links

Oxidative damage and antioxidant enzyme activities in experimental hypothyroidism.

Cell Biochem Funct. 2003 Dec;21(4):325-30.

PMID: 14624470 [PubMed - in process]

4

Peres CM, Homem De Bittencourt PJ Jr, Mendonca JR, Curi R.

Related Articles, Links

Evidence that macrophages transfer arachidonic acid and cholesterol to tissues in vivo.

Cell Biochem Funct. 2003 Dec;21(4):317-23.

PMID: 14624469 [PubMed - in process]

5

Aydemir EO, Var A, Uyanik BS, Ilkgul O, Aydede H, Sakarya A.

Related Articles, Links

The protective mechanisms of defibrotide on liver ischaemia-reperfusion injury.

Cell Biochem Funct. 2003 Dec;21(4):307-10.

PMID: 14624467 [PubMed - in process]

6

Cavalieri B, Perrelli MG, Aragno M, Ramadori P, Poli G, Cutrin JC.

Related Articles, Links

Ischaemic preconditioning modulates the activity of Kupffer cells during in vivo reperfusion injury of rat liver.

Cell Biochem Funct. 2003 Dec;21(4):299-305.

PMID: 14624466 [PubMed - in process]

7

Kim JO, Kim HN, Hwang MH, Shin HI, Kim SY, Park RW, Park EY, Kim IS, Van Wijnen AJ, Stein JL, Lian JB, Stein GS, Choi JY.

Related Articles, Links

Differential gene expression analysis using paraffin-embedded tissues after laser microdissection.

J Cell Biochem. 2003 Dec 1;90(5):998-1006.

PMID: 14624459 [PubMed - in process]

(a) List of matching pages for query “liver”.

1

Br J Surg. 2004 Nov 1 [Epub ahead of print]

Related Articles, Links

Radioisotope bone scans in the preoperative staging of hepatopancreatobiliary cancer.

Sheth H, Javed SS, Hilson AJ, Buscombe JR, Davidson BR.

University Department of Surgery, Royal Free Hospital, London, UK.

BACKGROUND: The aim of the study was to determine the value of radioisotope bone scans in the preoperative staging of patients with hepatopancreatobiliary (HPB) cancer. METHODS: Bone scanning was performed as part of a routine staging protocol in 402 consecutive patients with HPB cancer over a period of 5 years. Patients with positive bone scans underwent coned radiography, computed tomography with review on bone windows, or a bone biopsy. Bone scans were reviewed along with staging investigations, surgical and histological findings. Patients were followed for a minimum of 6 months. RESULTS: There were 171 patients with colorectal liver metastases, 106 with suspected pancreatic cancer, 47 with hepatocellular cancer, 52 with gallbladder cancer or cholangiocarcinoma, and 26 with other types of HPB cancer. Bone scans were negative in 377 patients (93.8 per cent) and positive in 25 patients (6.2 per cent). Of the 25 positive scans, 16 were falsely positive as a result of degenerative bone disease. Of nine patients with a true-positive bone scan, four had locally irresectable disease and four distant metastases. In only one patient did the bone scan result alone influence the decision to resect the HPB cancer. Overall sensitivity was 100 per cent, specificity 95.9 per cent, positive predictive value 36.0 per cent and negative predictive value 100 per cent. CONCLUSION: Bone scanning should not be included in the routine staging protocol for HPB cancer. Copyright (c) 2004 British Journal of Surgery Society Ltd. Published by John Wiley & Sons, Ltd.

PMID: 15521077 [PubMed - as supplied by publisher]

(b) The first matching page for “liver”.

Figure 2.3: Pages from the PubMed Web site.

well-defined structure, most textual databases provide a simple search interface where users type a list of keywords in a *single* search box (Figure 2.1). In contrast, a structured database often contains multi-attribute relational data (e.g., a book on the Amazon Web site may have the fields `title='Harry Potter'`, `author='J.K. Rowling'` and `isbn='0590353403'`) and supports *multi-attribute* search interfaces (Figure 2.2). Here, we will mainly focus on *textual databases* that support *single-attribute* keyword queries.

Typically, the users need to take the following steps in order to access pages in a Hidden-Web database:

1. **Step 1.** First, the user issues a query, say “liver,” through the search interface provided by the Web site (such as the one shown in Figure 2.1).
2. **Step 2.** Shortly after the user issues the query, she is presented with a *result index page*. That is, the Web site returns a list of links to potentially relevant Web pages, as shown in Figure 2.3(a).
3. **Step 3.** From the list in the result index page, the user identifies the pages that look “interesting” and follows the links. Clicking on a link leads the user to the actual Web page, such as the one shown in Figure 2.3(b), that the user wants to look at.

2.2.2 A generic Hidden-Web crawling algorithm

Given that the only “entry” to the pages in a Hidden-Web site is its search from, a Hidden-Web crawler should follow the three steps described in the previous section. That is, the crawler has to generate a query, issue it to the Web site, download the result index page, and follow the links to download the actual pages. In most cases, a crawler has limited time and network resources, so the crawler repeats these steps until it uses

Algorithm 2.2.1 *Crawling a Hidden-Web site*

Procedure

- (1) *while (there are available resources) do*
 // select a term to send to the site
- (2) $q_i = \text{SelectTerm}()$
 // send query and acquire result index page
- (3) $R(q_i) = \text{QueryWebSite}(q_i)$
 // download the pages of interest
- (4) $\text{Download}(R(q_i))$
- (5) *done*

Figure 2.4: Algorithm for crawling a Hidden-Web site.

up its resources.

In Figure 2.4 we show the generic algorithm for a Hidden-Web crawler. For simplicity, we assume that the Hidden-Web crawler issues single-term queries only.¹ The crawler first decides which query term it is going to use (Step (2)), issues the query, and retrieves the result index page (Step (3)). Finally, based on the links found on the result index page, it downloads the Hidden-Web pages from the site (Step (4)). This same process is repeated until all the available resources are used up (Step (1)).

Given this algorithm, we can see that the most critical decision that a crawler has to make is what query to issue next. If the crawler can issue successful queries that will return many matching pages, the crawler can finish its crawling early on using minimum resources. In contrast, if the crawler issues completely irrelevant queries

¹For most Web sites that assume “AND” for multi-keyword queries, single-term queries return the maximum number of results. Extending our work to multi-keyword queries is straightforward.

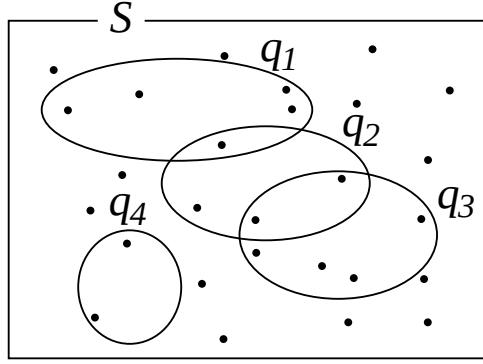


Figure 2.5: A set-formalization of the optimal query selection problem.

that do not return any matching pages, it may waste all of its resources simply issuing queries without ever retrieving actual pages. Therefore, how the crawler selects the next query can greatly affect its effectiveness. In the next section, we formalize this query selection problem.

2.2.3 Problem formalization

Theoretically, the problem of query selection can be formalized as follows: We assume that the crawler downloads pages from a Web site that has a set of pages S (the rectangle in Figure 2.5). We represent each Web page in S as a point (dots in Figure 2.5). Every potential query q_i that we may issue can be viewed as a subset of S , containing all the points (pages) that are returned when we issue q_i to the site. Each subset is associated with a weight that represents the cost of issuing the query. Under this formalization, our goal is to find which subsets (queries) cover the maximum number of points (Web pages) with the minimum total weight (cost). This problem is equivalent to the *set-covering* problem in graph theory [CLR01].

There are two main difficulties that we need to address in this formalization. First, in a practical situation, the crawler does not know which Web pages will be returned by

which queries, so the subsets of S are not known in advance. Without knowing these subsets the crawler cannot decide which queries to pick to maximize the coverage. Second, the set-covering problem is known to be NP-Hard [CLR01], so an efficient algorithm to solve this problem optimally in polynomial time has yet to be found.

In this dissertation, we will present an *approximation algorithm* that can find a near-optimal solution at a reasonable computational cost. Our algorithm leverages the observation that although we do not know which pages will be returned by each query q_i that we issue, we can *predict how many* pages will be returned. Based on this information our query selection algorithm can then select the “best” queries that cover the content of the Web site. We present our prediction method and our query selection algorithm in Section 2.3.

2.2.3.1 Performance Metric

Before we present our ideas for the query selection problem, we briefly discuss some of our notation and the cost/performance metrics.

Given a query q_i , we use $P(q_i)$ to denote the fraction of pages that we will get back if we issue query q_i to the site. For example, if a Web site has 10,000 pages in total, and if 3,000 pages are returned for the query $q_i = \text{medicine}$, then $P(q_i) = 0.3$. We use $P(q_1 \wedge q_2)$ to represent the fraction of pages that are returned from both q_1 and q_2 (i.e., the intersection of $P(q_1)$ and $P(q_2)$). Similarly, we use $P(q_1 \vee q_2)$ to represent the fraction of pages that are returned from either q_1 or q_2 (i.e., the union of $P(q_1)$ and $P(q_2)$).

We also use $Cost(q_i)$ to represent the cost of issuing the query q_i . Depending on the scenario, the cost can be measured either in time, network bandwidth, the number of interactions with the site, or it can be a function of all of these. As we will see later, our proposed algorithms are independent of the exact cost function.

In the most common case, the query cost consists of a number of factors, including the cost for submitting the query to the site, retrieving the result index page (Figure 2.3(a)) and downloading the actual pages (Figure 2.3(b)). We assume that submitting a query incurs a fixed cost of c_q . The cost for downloading the result index page is proportional to the number of matching documents to the query, while the cost c_d for downloading a matching document is also fixed. Then the overall cost of query q_i is

$$\text{Cost}(q_i) = c_q + c_r P(q_i) + c_d P(q_i). \quad (2.1)$$

In certain cases, some of the documents from q_i may have already been downloaded from previous queries. In this case, the crawler may skip downloading these documents and the cost of q_i can be

$$\text{Cost}(q_i) = c_q + c_r P(q_i) + c_d P_{\text{new}}(q_i). \quad (2.2)$$

Here, we use $P_{\text{new}}(q_i)$ to represent the fraction of the *new* documents from q_i that have *not* been retrieved from previous queries. Later in Section 2.3.1 we will study how we can estimate $P(q_i)$ and $P_{\text{new}}(q_i)$ to estimate the cost of q_i .

Since our algorithms are independent of the exact cost function, we will assume a generic cost function $\text{Cost}(q_i)$ here. When we need a concrete cost function, however, we will use Equation 2.2.

Given the notation, we can formalize the goal of a Hidden-Web crawler as follows:

Problem 2.2.1 *Find the set of queries $q_1, \dots, q_n$ that maximizes*

$$P(q_1 \vee \dots \vee q_n)$$

under the constraint

$$\sum_{i=1}^n \text{Cost}(q_i) \leq t.$$

Here, t is the maximum download resource that the crawler has.

2.3 Keyword Selection

How should a Hidden-Web crawler select the queries to issue? Given that the goal is to download the maximum number of unique documents from a textual database, we may consider one of the following options:

- *Random*: We select random keywords from, say, an English dictionary and issue them to the database. The hope is that a random query will return a reasonable number of matching documents.
- *Generic-frequency*: We analyze a generic document corpus collected elsewhere (say, from the Web) and obtain the generic frequency distribution of each keyword. Based on this generic distribution, we start with the most frequent keyword, issue it to the Hidden-Web database and retrieve the result. We then continue to the second-most frequent keyword and repeat this process until we exhaust all download resources. The hope is that the frequent keywords in a generic corpus will also be frequent in the Hidden-Web database, returning many matching documents.
- *Adaptive*: We analyze the documents returned from the previous queries issued to the Hidden-Web database and estimate which keyword is most likely to return the most documents. Based on this analysis, we issue the most “promising” query, and repeat the process.

Among these three general policies, we may consider the random policy as the *base comparison point* since it is expected to perform the worst. Between the generic-frequency and the adaptive policies, both policies may show similar performance if the crawled database has a *generic* document collection without a specialized topic. The adaptive policy, however, may perform significantly better than the generic-frequency policy if the database has a very specialized collection that is different from the generic corpus. We will experimentally compare these three policies in Section 2.4.

While the first two policies (random and generic-frequency policies) are easy to implement, we need to understand how we can analyze the downloaded pages to identify the most “promising” query in order to implement the adaptive policy. We address this issue in the rest of this section.

2.3.1 Estimating the number of matching pages

In order to identify the most promising query, we need to estimate how many new documents we will download if we issue the query q_i as the next query. That is, assuming that we have issued queries $q_1, \dots, q_{i-1}$ we need to estimate $P(q_1 \vee \dots \vee q_{i-1} \vee q_i)$, for every potential next query q_i and compare this value. In estimating this number, we note that we can rewrite $P(q_1 \vee \dots \vee q_{i-1} \vee q_i)$ as:

$$\begin{aligned}
& P((q_1 \vee \dots \vee q_{i-1}) \vee q_i) \\
&= P(q_1 \vee \dots \vee q_{i-1}) + P(q_i) - P((q_1 \vee \dots \vee q_{i-1}) \wedge q_i) \\
&= P(q_1 \vee \dots \vee q_{i-1}) + P(q_i) \\
&\quad - P(q_1 \vee \dots \vee q_{i-1})P(q_i | q_1 \vee \dots \vee q_{i-1})
\end{aligned} \tag{2.3}$$

In the above formula, note that we can precisely measure $P(q_1 \vee \dots \vee q_{i-1})$ and $P(q_i | q_1 \vee \dots \vee q_{i-1})$ by analyzing previously-downloaded pages: We know $P(q_1 \vee$

$\dots \vee q_{i-1}$), the union of all pages downloaded from $q_1, \dots, q_{i-1}$, since we have already issued $q_1, \dots, q_{i-1}$ and downloaded the matching pages.² We can also measure $P(q_i | q_1 \vee \dots \vee q_{i-1})$, the probability that q_i appears in the pages from $q_1, \dots, q_{i-1}$, by counting how many times q_i appears in the pages from $q_1, \dots, q_{i-1}$. Therefore, we only need to estimate $P(q_i)$ to evaluate $P(q_1 \vee \dots \vee q_i)$. We may consider a number of different ways to estimate $P(q_i)$, including the following:

1. *Independence estimator*: We assume that the appearance of the term q_i is independent of the terms $q_1, \dots, q_{i-1}$. That is, we assume that $P(q_i) = P(q_i | q_1 \vee \dots \vee q_{i-1})$.
2. *Zipf estimator*: In [IG02], Ipeirotis et al. proposed a method to estimate how many times a particular term occurs in the *entire* corpus based on a subset of documents from the corpus. Their method exploits the fact that the frequency of terms inside text collections follows a power law distribution [Zip49, Man]. That is, if we rank all terms based on their occurrence frequency (with the most frequent term having a rank of 1, second most frequent a rank of 2 etc.), then the frequency f of a term inside the text collection is given by:

$$f = \alpha(r + \beta)^{-\gamma} \tag{2.4}$$

where r is the rank of the term and α, β , and γ are constants that depend on the text collection.

We illustrate how we can use the above equation to estimate the $P(q_i)$ values through the following example:

²For exact estimation, we need to know the total number of pages in the site. However, in order to compare only relative values among queries, this information is not actually needed.

Example 2.3.1 Assume that we have already issued queries $q_1 = \text{disk}$, $q_2 = \text{java}$, $q_3 = \text{computer}$ to a text database. The database contains 100,000 documents in total, and for each query q_1 , q_2 , and q_3 , the database returned 2,000, 1,100, and 6,000 pages, respectively. We show $P(q_i)$ for each query in the second column of Figure 2.6(a). For example, $P(\text{computer}) = \frac{6,000}{100,000} = 0.06$. We assume that the documents retrieved from q_1 , q_2 and q_3 do not have a significant overlap, and we downloaded a total of 8,000 unique documents from the three queries. Given this, $P(\text{computer} | q_1 \vee q_2 \vee q_3) = \frac{6,000}{8,000} = 0.75$, etc. We show $P(q_i | q_1 \vee q_2 \vee q_3)$ for every query in the third column of Figure 2.6(a).

Within the retrieved documents, we can count how many times other terms t_k 's appear (i.e., $P(t_k | q_1 \vee q_2 \vee q_3)$) and we also show these numbers in the third column. For example, **data** appears in 3,200 documents out of 8,000 retrieved documents, so $P(\text{data} | q_1 \vee q_2 \vee q_3) = 0.4$. However, we do not know exactly how many times the term **data** appears in the entire database, so the value of $P(\text{data})$ is unknown. We use question marks to represent unknown values. Our goal is to estimate these unknown values.

For estimation, we first rank each term t_k based on its $P(t_k | q_1 \vee q_2 \vee q_3)$ value. The rank of each term is shown in the fourth column of Figure 2.6(a). For example, the term **computer** appears most frequently within the downloaded pages, so its rank is 1.

Now, given Equation 2.4, we know that the rank $R(t_k)$ and the frequency $P(t_k)$ of a term roughly satisfies the following equation:

$$P(t_k) = \alpha [R(t_k) + \beta]^{-\gamma}. \quad (2.5)$$

Then using the known $P(t_k)$ values from previous queries, we can compute the constants α , β , and γ . That is, given the $P(\text{computer})$, $P(\text{disk})$, and $P(\text{java})$ values and their ranks, we can find the best-fitting curve for Equation 2.4(b).

This curve fitting results in the following constants: $\alpha = 0.08$, $\beta = 0.25$, $\gamma = 1.15$. In Figure 2.6(b), we show the fitted curve together with the rank of each term.

Once we have obtained these constants, we can estimate all unknown $P(t_k)$ values. For example, since $R(data)$ is 2, we can estimate that

$$P(data) = \alpha[R(data) + \beta]^{-\gamma} \quad (2.6)$$

$$= 0.08(2 + 0.25)^{-1.15} = 0.031. \quad (2.7)$$

After we estimate $P(q_i)$ and $P(q_i|q_1 \vee \dots \vee q_{i-1})$ values, we can calculate $P(q_1 \vee \dots \vee q_i)$. In Section 2.3.3, we explain how we can efficiently compute $P(q_i|q_1 \vee \dots \vee q_{i-1})$ by maintaining a succinct summary table. In the next section, we first examine how we can use this value to decide which query we should issue next to the Hidden-Web site.

2.3.2 Query selection algorithm

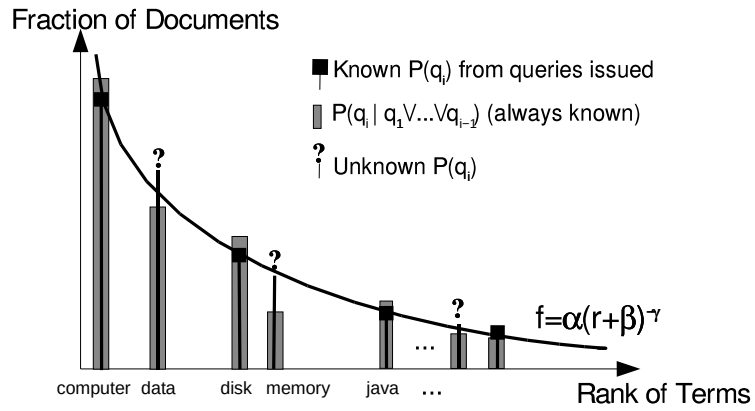
The goal of the Hidden-Web crawler is to download the maximum number of unique documents from a database using its limited download resources. Given this goal, the Hidden-Web crawler has to take two factors into account. (1) the number of new documents that can be obtained from the query q_i and (2) the cost of issuing the query q_i . For example, if two queries, q_i and q_j , incur the same cost, but q_i returns more new pages than q_j , q_i is more desirable than q_j . Similarly, if q_i and q_j return the same number of new documents, but q_i incurs less cost than q_j , q_i is more desirable. Based on this observation, the Hidden-Web crawler may use the following *efficiency* metric to quantify the desirability of the query q_i :

$$Efficiency(q_i) = \frac{P_{new}(q_i)}{Cost(q_i)} \quad (2.8)$$

$$q_1 = \text{disk}, q_2 = \text{java}, q_3 = \text{computer}$$

Term t_k	$P(t_k)$	$P(t_k q_1 \vee q_2 \vee q_3)$	$R(t_k)$
computer	0.06	0.75	1
data	?	0.4	2
disk	0.02	0.25	3
memory	?	0.2	4
java	0.011	0.13	5
...	...	...	...

(a) Probabilities of terms after queries q_1, q_2, q_3 .



(b) Zipf curve of example data.

Figure 2.6: Fitting a Zipf Distribution in order to estimate $P(q_i)$.

Algorithm 2.3.1 *Greedy SelectTerm()****Parameters:***

T: The list of potential query keywords

Procedure

- (1) *Foreach* t_k *in* T *do*
- (2) *Estimate* $\text{Efficiency}(t_k) = \frac{P_{new}(t_k)}{\text{Cost}(t_k)}$
- (3) *done*
- (4) *return* t_k *with maximum* $\text{Efficiency}(t_k)$

Figure 2.7: Algorithm for selecting the next query term.

Here, $P_{new}(q_i)$ represents the amount of *new* documents returned for q_i (the pages that have not been returned for previous queries). $\text{Cost}(q_i)$ represents the cost of issuing the query q_i .

Intuitively, the efficiency of q_i measures how many new documents are retrieved per unit cost, and can be used as an indicator of how well our resources are spent when issuing q_i . Thus, the Hidden-Web crawler can estimate the efficiency of every candidate q_i , and select the one with the highest value. By using its resources more efficiently, the crawler may eventually download the maximum number of unique documents. In Figure 2.7, we show the query selection function that uses the concept of efficiency. In principle, this algorithm takes a greedy approach and tries to maximize the “potential gain” in every step.

We can estimate the efficiency of every query using the estimation method described in Section 2.3.1. That is, the size of the new documents from the query q_i ,

$P_{new}(q_i)$, is

$$P_{new}(q_i) \tag{2.9}$$

$$= P(q_1 \vee \dots \vee q_{i-1} \vee q_i) - P(q_1 \vee \dots \vee q_{i-1}) \tag{2.10}$$

$$= P(q_i) - P(q_1 \vee \dots \vee q_{i-1})P(q_i|q_1 \vee \dots \vee q_{i-1}) \tag{2.11}$$

from Equation 2.3, where $P(q_i)$ can be estimated using one of the methods described in section 2.3. We can also estimate $Cost(q_i)$ similarly. For example, if $Cost(q_i)$ is

$$Cost(q_i) = c_q + c_r P(q_i) + c_d P_{new}(q_i) \tag{2.12}$$

(Equation 2.2), we can estimate $Cost(q_i)$ by estimating $P(q_i)$ and $P_{new}(q_i)$.

2.3.3 Efficient calculation of query statistics

In estimating the efficiency of queries, we found that we need to measure $P(q_i|q_1 \vee \dots \vee q_{i-1})$ for every potential query q_i . This calculation can be very time-consuming if we repeat it from scratch for every query q_i in every iteration of our algorithm. In this section, we explain how we can compute $P(q_i|q_1 \vee \dots \vee q_{i-1})$ efficiently by maintaining a small table that we call a *query statistics table*.

The main idea for the query statistics table is that $P(q_i|q_1 \vee \dots \vee q_{i-1})$ can be measured by counting how many times the keyword q_i appears within the documents downloaded from $q_1, \dots, q_{i-1}$. We record these counts in a table, as shown in Figure 2.8(a). The left column of the table contains all potential query terms and the right column contains the number of previously-downloaded documents containing the respective term. For example, the table in Figure 2.8(a) shows that we have downloaded 50 documents so far, and the term `model` appears in 10 of these documents. Given this number, we can compute that $P(model|q_1 \vee \dots \vee q_{i-1}) = \frac{10}{50} = 0.2$.

We note that the query statistics table needs to be updated whenever we issue a

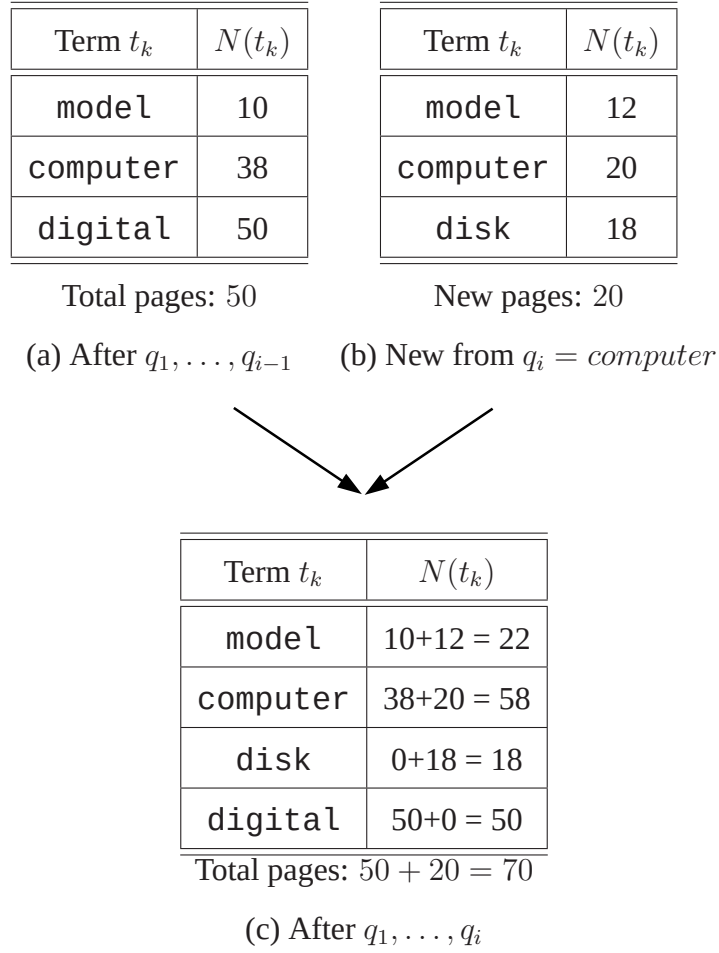


Figure 2.8: Updating the query statistics table.

new query q_i and download more documents. This update can be done efficiently as we illustrate in the following example.

Example 2.3.2 After examining the query statistics table of Figure 2.8(a), we have decided to use the term *computer* as our next query q_i . From the new query $q_i = \text{computer}$, we downloaded 20 more new pages. Out of these, 12 contain the keyword *model* and 18 the keyword *disk*. The table in Figure 2.8(b) shows the frequency of each term in the newly-downloaded pages.

We can update the old table (Figure 2.8(a)) to include this new information by

simply adding corresponding entries in Figures 2.8(a) and (b). The result is shown on Figure 2.8(c). For example, the keyword *model* exists in $10 + 12 = 22$ pages within the pages retrieved from $q_1, \dots, q_i$. According to this new table, $P(model|q_1 \vee \dots \vee q_i)$ is now $\frac{22}{70} = 0.3$.

2.3.4 Crawling sites that limit the number of results

In certain cases, when a query matches a large number of pages, the Hidden-Web site returns only a portion of those pages. For example, the Open Directory Project [Dmo] allows the users to see only up to 10,000 results after they issue a query.

Obviously, this kind of limitation has an immediate effect on our Hidden-Web crawler. First, since we can only retrieve up to a specific number of pages per query, our crawler will need to issue more queries (and potentially will use up more resources) in order to download all the pages. Second, the query selection method that we presented in Section 2.3.2 assumes that for every potential query q_i , we can find $P(q_i|q_1 \vee \dots \vee q_{i-1})$. That is, for every query q_i we can find the fraction of documents in the *whole* text database that contains q_i with at least one of $q_1, \dots, q_{i-1}$.

However, if the text database returned only a portion of the results for any of the $q_1, \dots, q_{i-1}$ then the value $P(q_i|q_1 \vee \dots \vee q_{i-1})$ is not accurate and may affect our decision for the next query q_i , and potentially the performance of our crawler. Since we cannot retrieve more results per query than the maximum number the Web site allows, our crawler has no other choice besides submitting more queries. However, there is a way to *estimate* the correct value for $P(q_i|q_1 \vee \dots \vee q_{i-1})$ in the case where the Web site returns only a portion of the results.

Again, assume that the Hidden-Web site we are currently crawling is represented as the rectangle on Figure 2.9 and its pages as points in the figure. Assume that we have already issued queries $q_1, \dots, q_{i-1}$ which returned a number of results less than

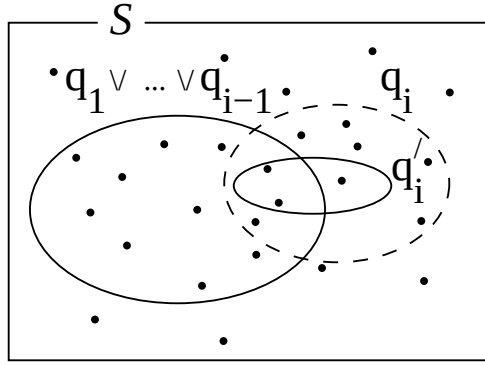


Figure 2.9: A Web site that does not return all the results.

the maximum number than the site allows, and therefore we have downloaded all the pages for these queries (big circle in Figure 2.9). That is, at this point, our estimation for $P(q_i | q_1 \vee \dots \vee q_{i-1})$ is accurate. Now assume that we submit query q_i to the Web site, but due to a limitation in the number of results that we get back, we retrieve the set q'_i (small circle in Figure 2.9) instead of the set q_i (dashed circle in Figure 2.9). Now we need to update our *query statistics table* so that it has accurate information for the next step. That is, although we got the set q'_i back, for every potential query q_{i+1} we need to find $P(q_{i+1} | q_1 \vee \dots \vee q_i)$:

$$\begin{aligned}
 &P(q_{i+1} | q_1 \vee \dots \vee q_i) \\
 &= \frac{1}{P(q_1 \vee \dots \vee q_i)} \cdot [P(q_{i+1} \wedge (q_1 \vee \dots \vee q_{i-1})) + \\
 &\quad P(q_{i+1} \wedge q_i) - P(q_{i+1} \wedge q_i \wedge (q_1 \vee \dots \vee q_{i-1}))] \tag{2.13}
 \end{aligned}$$

In the previous equation, we can find $P(q_1 \vee \dots \vee q_i)$ by estimating $P(q_i)$ with the method shown in Section 2.3. Additionally, we can calculate $P(q_{i+1} \wedge (q_1 \vee \dots \vee q_{i-1}))$ and $P(q_{i+1} \wedge q_i \wedge (q_1 \vee \dots \vee q_{i-1}))$ by directly examining the documents that we have

downloaded from queries $q_1, \dots, q_{i-1}$. The term $P(q_{i+1} \wedge q_i)$ however is unknown and we need to estimate it. Assuming that q'_i is a random sample of q_i , then:

$$\frac{P(q_{i+1} \wedge q_i)}{P(q_{i+1} \wedge q'_i)} = \frac{P(q_i)}{P(q'_i)} \quad (2.14)$$

From Equation 2.14 we can calculate $P(q_{i+1} \wedge q_i)$ and after we replace this value to Equation 2.13 we can find $P(q_{i+1} | q_1 \vee \dots \vee q_i)$.

2.4 Experimental Evaluation

In this section we experimentally evaluate the performance of the various algorithms for Hidden-Web crawling presented in this dissertation. Our goal is to validate our theoretical analysis through real-world experiments, by crawling popular Hidden-Web sites of textual databases. Since the number of documents that are discovered and downloaded from a textual database depends on the selection of the words that will be issued as queries³ to the search interface of each site, we compare the various selection policies that were described in section 2.3, namely the *random*, *generic-frequency*, and *adaptive* algorithms.

The *adaptive* algorithm learns new keywords and terms from the documents that it downloads, and its selection process is driven by a cost model as described in Section 2.3.2. To keep our experiment and its analysis simple at this point, we will assume that the cost for every query is constant. That is, our goal is to maximize the number of downloaded pages by issuing the least number of queries. Later, in Section 2.4.4 we will present a comparison of our policies based on a more elaborate cost model. In addition, we use the *independence estimator* (Section 2.3.1) to estimate $P(q_i)$ from

³Throughout our experiments, once an algorithm has submitted a query to a database, we exclude the query from subsequent submissions to the same database from the same algorithm.

downloaded pages. Although the *independence estimator* is a simple estimator, our experiments will show that it can work very well in practice.

For the *generic-frequency* policy, we compute the frequency distribution of words that appear in a 5.5-million-Web-page corpus downloaded from 154 Web sites of various topics [NCO04]. Keywords are selected based on their decreasing frequency with which they appear in this document set, with the most frequent one being selected first, followed by the second-most frequent keyword, etc.⁴

Regarding the *random* policy, we use the same set of words collected from the Web corpus, but in this case, instead of selecting keywords based on their relative frequency, we choose them randomly (uniform distribution). In order to further investigate how the quality of the potential query-term list affects the random-based algorithm, we construct two sets: one with the 16,000 most frequent words of the term collection used in the generic-frequency policy (hereafter, the random policy with the set of 16,000 words will be referred to as *random-16K*), and another set with the 1 million most frequent words of the same collection as above (hereafter, referred to as *random-1M*). The former set has frequent words that appear in a large number of documents (at least 10,000 in our collection), and therefore can be considered of “high-quality” terms. The latter set though contains a much larger collection of words, among which some might be bogus, and meaningless.

The experiments were conducted by employing each one of the aforementioned algorithms (adaptive, generic-frequency, random-16K, and random-1M) to crawl and download contents from three Hidden-Web sites: The PubMed Medical Library [Pub], Amazon [Ama], and the Open Directory Project[Dmo]. According to the information on *PubMed*’s Web site, its collection contains approximately 14 million abstracts of biomedical articles. We consider these abstracts as the “documents” in the site, and in

⁴We did not manually exclude *stop words* (e.g., the, is, of, etc.) from the keyword list. As it turns out, all Web sites except PubMed return matching documents for the stop words, such as *the*.

each iteration of the adaptive policy, we use these abstracts as input to the algorithm. Thus our goal is to “discover” as many unique abstracts as possible by repeatedly querying the Web query interface provided by PubMed. The Hidden-Web crawling on the PubMed Web site can be considered as topic-specific, due to the fact that all abstracts within PubMed are related to the fields of medicine and biology.

In the case of the *Amazon* Web site, we are interested in downloading all the hidden pages that contain information on books. The querying to Amazon is performed through the Software Developer’s Kit that Amazon provides for interfacing to its Web site, and which returns results in XML form. The generic “keyword” field is used for searching, and as input to the adaptive policy we extract the product description and the text of customer reviews when present in the XML reply. Since Amazon does not provide any information on how many books it has in its catalogue, we use random sampling on the 10-digit ISBN number of the books to estimate the size of the collection. Out of the 10,000 random ISBN numbers queried, 46 are found in the Amazon catalogue, therefore the size of its book collection is estimated to be $\frac{46}{10000} \cdot 10^{10} = 4.6$ million books. It’s also worth noting here that Amazon poses an upper limit on the number of results (books in our case) returned by each query, which is set to 32,000.

As for the third Hidden-Web site, the *Open Directory Project* (hereafter also referred to as *dmoz*), the site maintains the links to 3.8 million sites together with a brief summary of each listed site. The links are searchable through a keyword-search interface. We consider each indexed link together with its brief summary as the document of the *dmoz* site, and we provide the short summaries to the adaptive algorithm to drive the selection of new keywords for querying. On the *dmoz* Web site, we perform two Hidden-Web crawls: the first is on its generic collection of 3.8-million indexed sites, regardless of the category that they fall into. The other crawl is performed specifically on the Arts section of *dmoz* (<http://dmoz.org/Arts>), which comprises

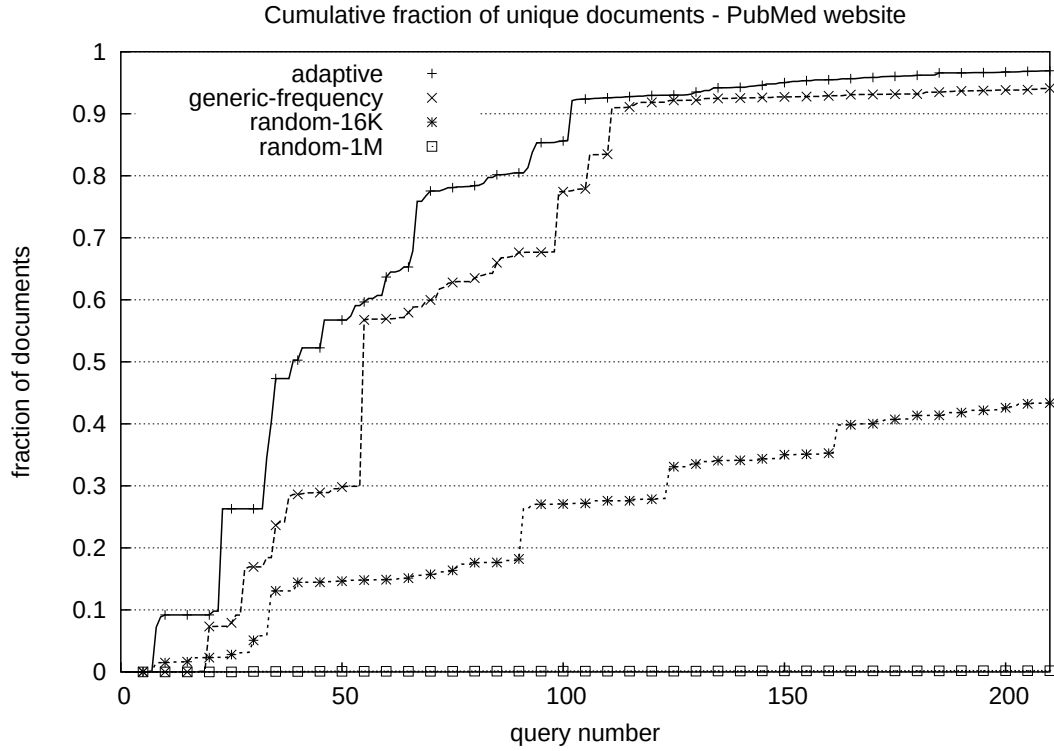


Figure 2.10: Coverage of policies for Pubmed

of approximately 429,000 indexed sites that are relevant to Arts, making this crawl topic-specific, as in PubMed. Like Amazon, *dmoz* also enforces an upper limit on the number of returned results, which is 10,000 links with their summaries.

2.4.1 Comparison of policies

The first question that we seek to answer is the evolution of the *coverage* metric as we submit queries to the sites. That is, what fraction of the collection of documents stored in the Hidden-Web site can we download as we continuously query for new words selected using the policies described above? More formally, we are interested in the value of $P(q_1 \vee \dots \vee q_{i-1} \vee q_i)$, after we submit $q_1, \dots, q_i$ queries, and as i increases.

In Figures 2.10, 2.11, 2.12, and 2.13 we present the coverage metric for each pol-

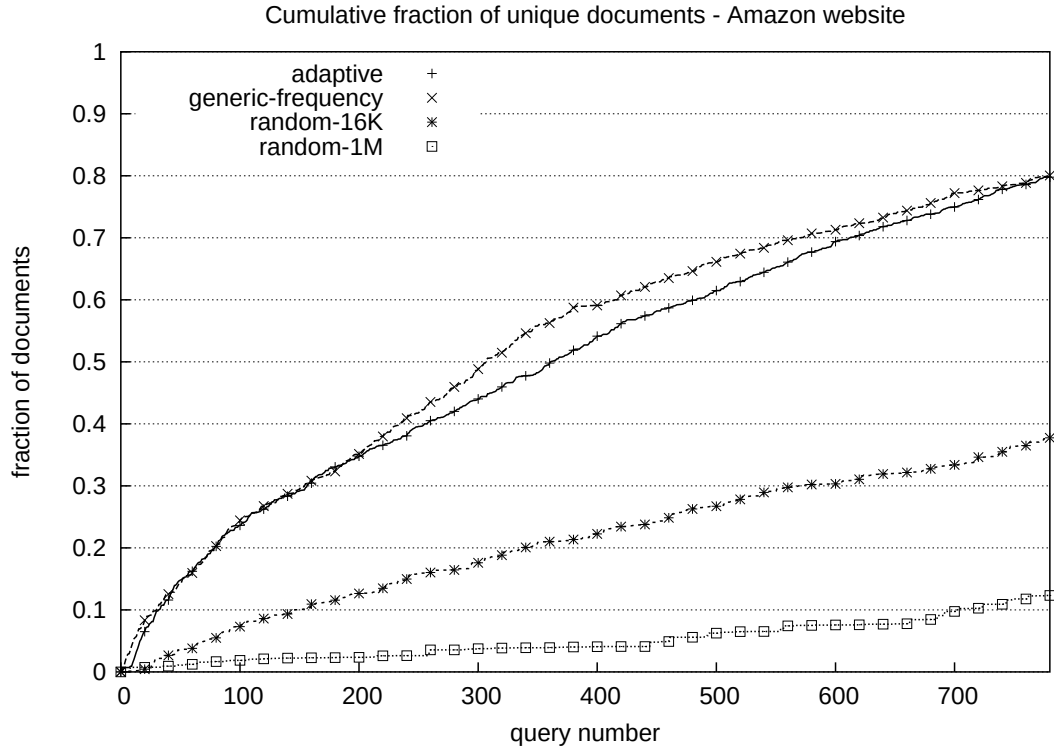


Figure 2.11: Coverage of policies for Amazon

icity, as a function of the query number, for the Web sites of PubMed, Amazon, general *dmoz* and the art-specific *dmoz*, respectively. On the y-axis the fraction of the total documents downloaded from the website is plotted, while the x-axis represents the query number. A first observation from these graphs is that in general, the generic-frequency and the adaptive policies perform much better than the random-based algorithms. In all of the figures, the graphs for the random-1M and the random-16K are significantly below those of other policies.

Between the generic-frequency and the adaptive policies, we can see that the latter outperforms the former when the site is topic specific. For example, for the PubMed site (Figure 2.10), the adaptive algorithm issues only 83 queries to download almost 80% of the documents stored in PubMed, but the generic-frequency algorithm requires 106 queries for the same coverage,. For the *dmoz*/Arts crawl (Figure 2.13), the differ-

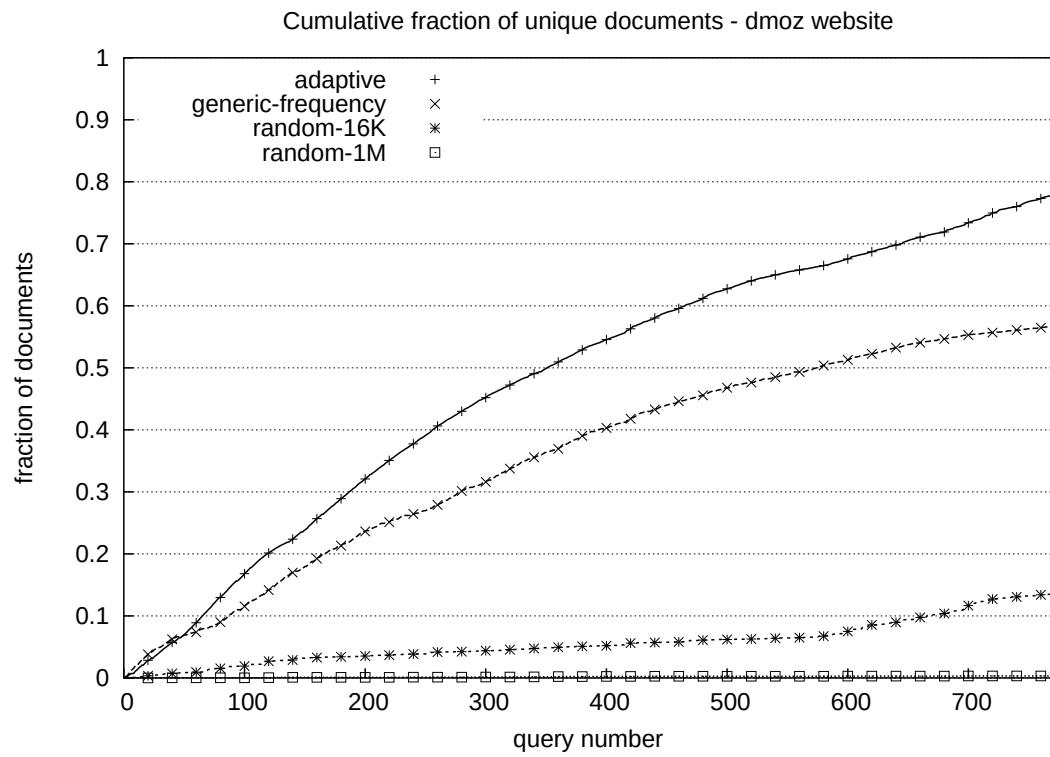


Figure 2.12: Coverage of policies for general dmoz

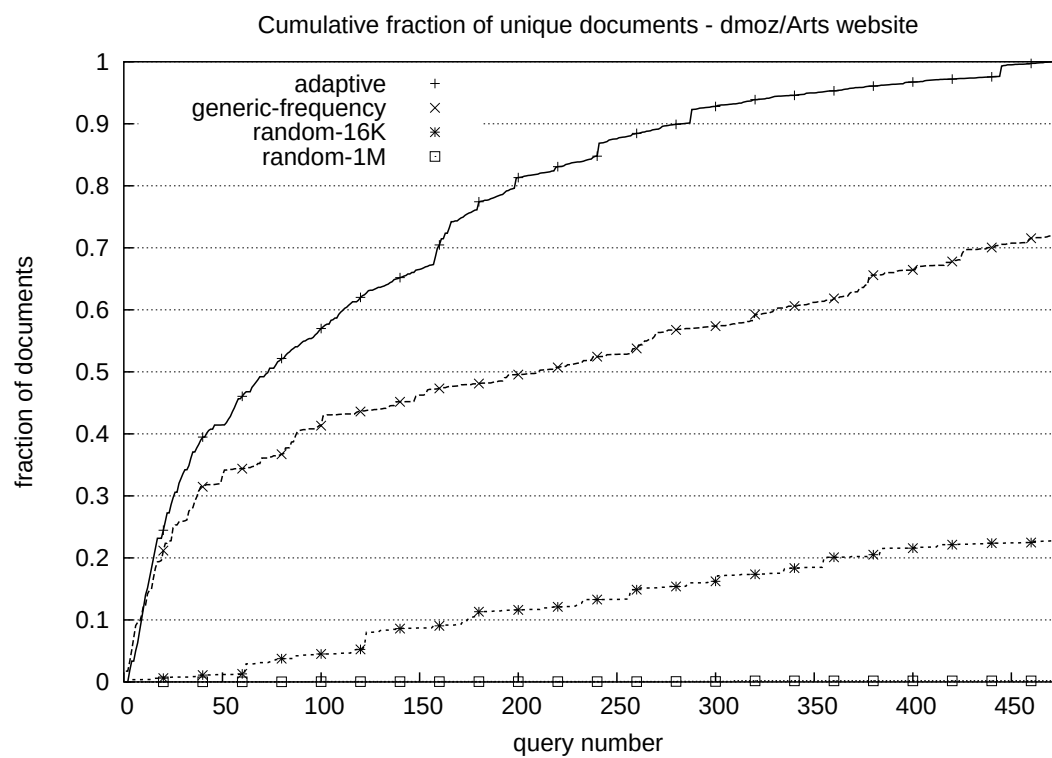


Figure 2.13: Coverage of policies for the Arts section of dmoz

ence is even more substantial: the adaptive policy is able to download 99.98% of the total sites indexed in the Directory by issuing 471 queries, while the frequency-based algorithm is much less effective using the same number of queries, and discovers only 72% of the total number of indexed sites. The adaptive algorithm, by examining the contents of the pages that it downloads at each iteration, is able to identify the topic of the site as expressed by the words that appear most frequently in the result-set. Consequently, it is able to select words for subsequent queries that are more relevant to the site, than those preferred by the generic-frequency policy, which are drawn from a large, generic collection. Table 2.1 shows a sample of 10 keywords out of 211 chosen and submitted to the PubMed Web site by the adaptive algorithm, but not by the other policies. For each keyword, we present the number of the iteration, along with the number of results that it returned. As one can see from the table, these keywords are highly relevant to the topics of medicine and biology of the Public Medical Library, and match against numerous articles stored in its Web site.

In both cases examined in Figures 2.10, and 2.13, the random-based policies perform much worse than the adaptive algorithm, and the generic-frequency. It is worthy noting however, that the random-based policy with the small, carefully selected set of 16,000 “quality” words manages to download a considerable fraction of 42.5% from the PubMed Web site after 200 queries, while the coverage for the Arts section of dmoz reaches 22.7%, after 471 queried keywords. On the other hand, the random-based approach that makes use of the vast collection of 1 million words, among which a large number is bogus keywords, fails to download even a mere 1% of the total collection, after submitting the same number of query words.

For the generic collections of Amazon and the *dmoz* sites, shown in Figures 2.11 and 2.12 respectively, we get mixed results: The generic-frequency policy shows slightly better performance than the adaptive policy for the Amazon site (Figure 2.11),

<i>Iteration</i>	<i>Keyword</i>	<i>Number of Results</i>
23	department	2,719,031
34	patients	1,934,428
53	clinical	1,198,322
67	treatment	4,034,565
69	medical	1,368,200
70	hospital	503,307
146	disease	1,520,908
172	protein	2,620,938
174	molecular	951,639
185	diagnosis	4,276,518

Table 2.1: Sample of keywords queried to PubMed exclusively by the adaptive policy

and the adaptive method clearly outperforms the generic-frequency for the general *dmoz* site (Figure 2.12). A closer look at the log files of the two Hidden-Web crawlers reveals the main reason: Amazon was functioning in a very flaky way when our adaptive crawler visited it: for certain queries a large number of results would result in errors and could not be retrieved from Amazon’s Web site. Thus, we suspect that the slightly poor performance of the adaptive policy is due to this experimental variance. At the same time, this observation indicates that, in practice, when implementing a real system for performing Hidden-Web crawling, it is very important to be able to handle such unpredictable situations as they can significantly affect the performance of our Hidden-Web crawler. Aside from this experimental variance, the Amazon result indicates that if the collection and the words that a Hidden-Web site contains are generic enough, then the generic-frequency approach may be a good candidate algorithm for effective crawling.

As in the case of topic-specific Hidden-Web sites, the random-based policies also exhibit poor performance compared to the other two algorithms when crawling generic sites: for the Amazon Web site, random-16K succeeds in downloading almost 36.7% after issuing 775 queries, alas for the generic collection of dmoz, the fraction of the collection of links downloaded is 13.5% after the 770th query. Finally, as expected, random-1M is even worse than random-16K, downloading only 14.5% of Amazon and 0.3% of the generic dmoz.

In summary, the adaptive algorithm performs remarkably well in all cases: it is able to discover and download most of the documents stored in Hidden-Web sites by issuing the least number of queries. When the collection refers to a specific topic, it is able to identify the keywords most relevant to the topic of the site and consequently ask for terms that is most likely that will return a large number of results . On the other hand, the generic-frequency policy proves to be quite effective too, though less than the adaptive: it is able to retrieve relatively fast a large portion of the collection, and when the site is not topic-specific, its effectiveness can reach that of adaptive (e.g. Amazon). Finally, the random policy performs poorly in general, and should not be preferred.

2.4.2 Impact of the initial query

An interesting issue that deserves further examination is whether the initial choice of the keyword used as the first query issued by the adaptive algorithm affects its effectiveness in subsequent iterations. The choice of this keyword is not done by the selection of the adaptive algorithm itself and has to be manually set, since its query statistics tables have not been populated yet. Thus, the selection is generally arbitrary, so for purposes of fully automating the whole process, some additional investigation seems necessary.

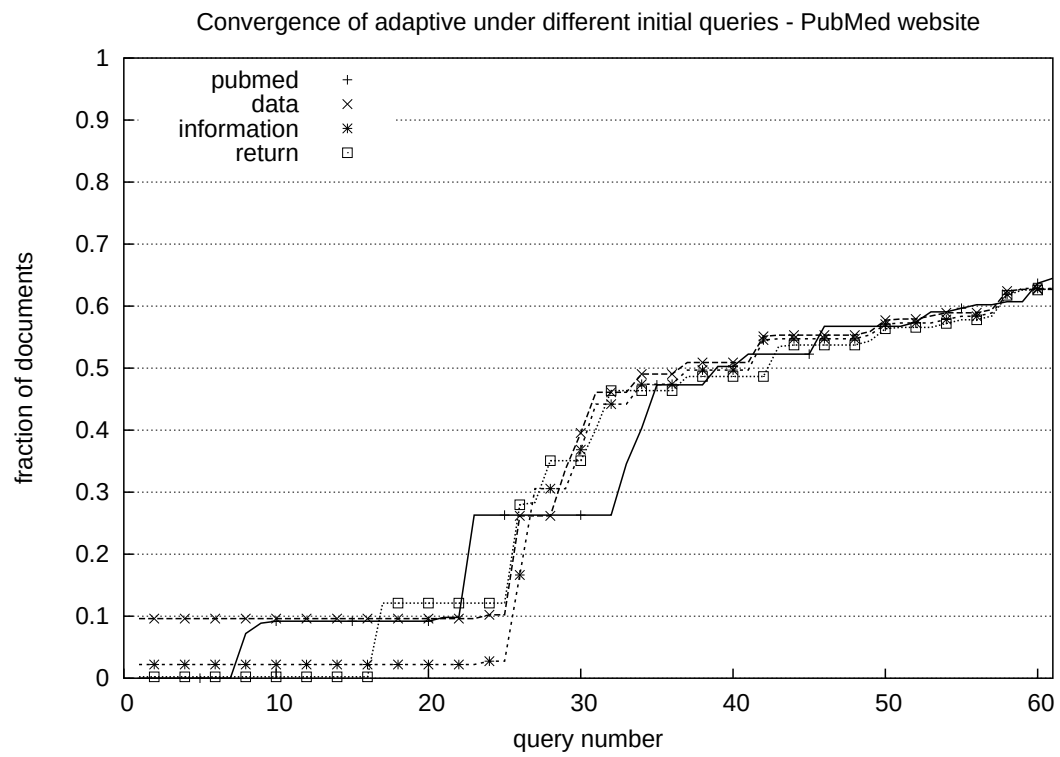


Figure 2.14: Convergence of the adaptive algorithm using different initial queries for crawling the PubMed Web site

For this reason, we initiated three adaptive Hidden-Web crawlers targeting the PubMed Web site with different seed-words: the word `data`, which returns 1,344,999 results, the word “information” that reports 308,474 documents, and the word `return` that retrieves 29,707 pages, out of 14 million. These keywords represent varying degrees of term popularity in PubMed, with the first one being of high popularity, the second of medium, and the third of low. We also show results for the keyword `pubmed`, used in the experiments for coverage of Section 2.4.1, and which returns 695 articles. As we can see from Figure 2.14, after a small number of queries, all four crawlers roughly download the same fraction of the collection, regardless of their starting point: Their coverages are roughly equivalent from the 25th query. Eventually, all four crawlers use the same set of terms for their queries, regardless of the initial query. In the specific experiment, from the 36th query onward, all four crawlers use the same terms for their queries in each iteration, or the same terms are used off by one or two query numbers. Our result confirms the observation of [CC01] that the choice of the initial query has minimal effect on the final performance. We can explain this intuitively as follows: Our algorithm approximates the optimal set of queries to use for a particular Web site. Once the algorithm has issued a significant number of queries, it has an accurate estimation of the content of the Web site, regardless of the initial query. Since this estimation is similar for all runs of the algorithm, the crawlers will use roughly the same queries.

2.4.3 Impact of the limit in the number of results

While the Amazon and dmoz sites have the respective limit of 32,000 and 10,000 in their result sizes, these limits may be larger than those imposed by other Hidden-Web sites. In order to investigate how a “tighter” limit in the result size affects the performance of our algorithms, we performed two additional crawls to the generic-

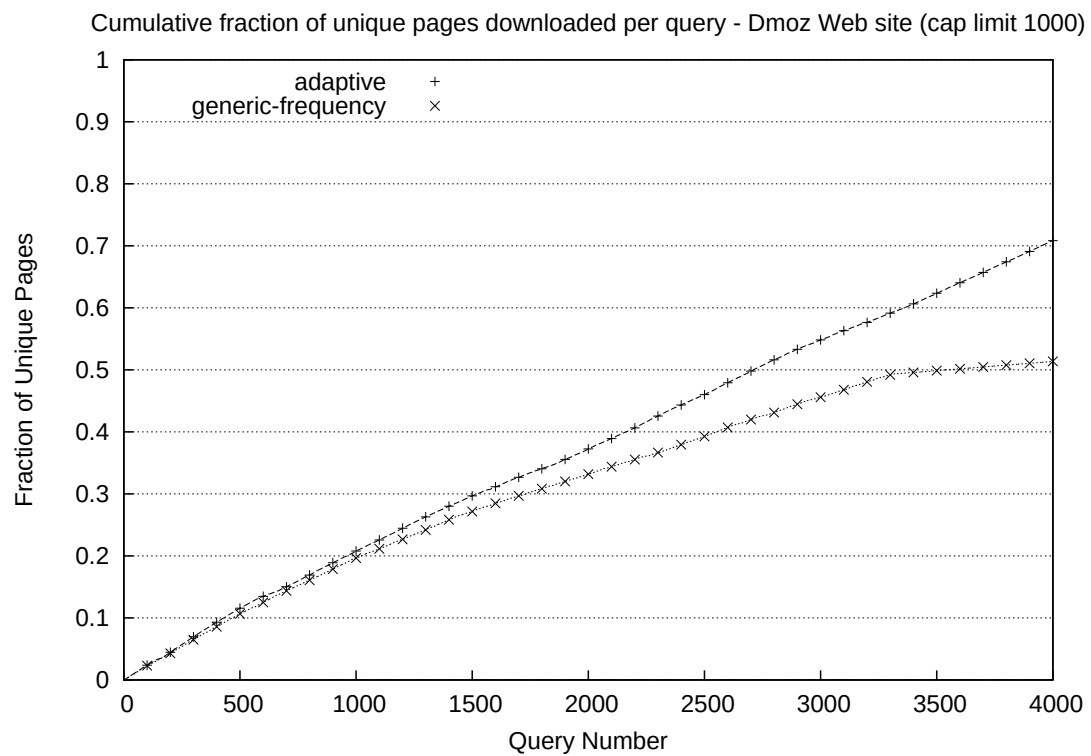


Figure 2.15: Coverage of general dmoz after limiting the number of results to 1,000

dmoz site: we ran the generic-frequency and adaptive policies but we retrieved only up to the top 1,000 results for every query. In Figure 2.15 we plot the coverage for the two policies as a function of the number of queries. As one might expect, by comparing the new result in Figure 2.15 to that of Figure 2.12 where the result limit was 10,000, we conclude that the tighter limit requires a higher number of queries to achieve the same coverage. For example, when the result limit was 10,000, the adaptive policy could download 70% of the site after issuing 630 queries, while it had to issue about 4,000 queries to download 70% of the site when the limit was 1,000. On the other hand, our new result shows that even with a tight result limit, it is still possible to download most of a Hidden-Web site after issuing a reasonable number of queries. The adaptive policy could download slightly more than 70% of the site after issuing 4,000 queries when the limit was 1,000. Finally, our result shows that our adaptive policy consistently outperforms the generic-frequency policy regardless of the result limit. In both Figure 2.15 and Figure 2.12, our adaptive policy shows significantly larger coverage than the generic-frequency policy for the same number of queries.

2.4.4 Incorporating the document download cost

For brevity of presentation, the performance evaluation results provided so far assumed a simplified cost-model where every query involved a constant cost. In this section we present results regarding the performance of the adaptive and generic-frequency algorithms using Equation 2.2 to drive our query selection process. As we discussed in Section 2.2.3.1, this query cost model includes the cost for submitting the query to the site, retrieving the result index page, and also downloading the actual pages. For these costs, we examined the size of every result in the index page and the sizes of the documents, and we chose $c_q = 100$, $c_r = 100$, and $c_d = 10000$, as values for the parameters of Equation 2.2, and for the particular experiment that we ran on the

PubMed website. The values that we selected imply that the cost for issuing one query and retrieving one result from the result index page are roughly the same, while the cost for downloading an actual page is 100 times larger. We believe that these values are reasonable for the PubMed Web site.

Figure 2.16 shows the coverage of the adaptive and generic- frequency algorithms as a function of the resource units used during the download process. The horizontal axis is the amount of resources used, and the vertical axis is the coverage. As it is evident from the graph, the adaptive policy makes more efficient use of the available resources, as it is able to download more articles than the generic-frequency, using the same amount of resource units. However, the difference in coverage is less dramatic in this case, compared to the graph of Figure 2.10. The smaller difference is due to the fact that under the current cost metric, the download cost of documents constitutes a significant portion of the cost. Therefore, when both policies downloaded the same number of documents, the saving of the adaptive policy is not as dramatic as before. That is, the savings in the query cost and the result index download cost is only a relatively small portion of the overall cost. Still, we observe noticeable savings from the adaptive policy. At the total cost of 8000, for example, the coverage of the adaptive policy is roughly 0.5 while the coverage of the frequency policy is only 0.3.

2.4.5 Other observations

Another interesting side-effect of the adaptive algorithm is the support for multilingual Hidden-Web sites, without any additional modification. In our attempt to explain the significantly better performance of the adaptive algorithm for the generic dmoz Web site (which is not topic-specific), we noticed that the adaptive algorithm issues non-English queries to the site. This is because the adaptive algorithm learns its vocabulary from the pages it downloads, therefore is able to discover frequent words that do not

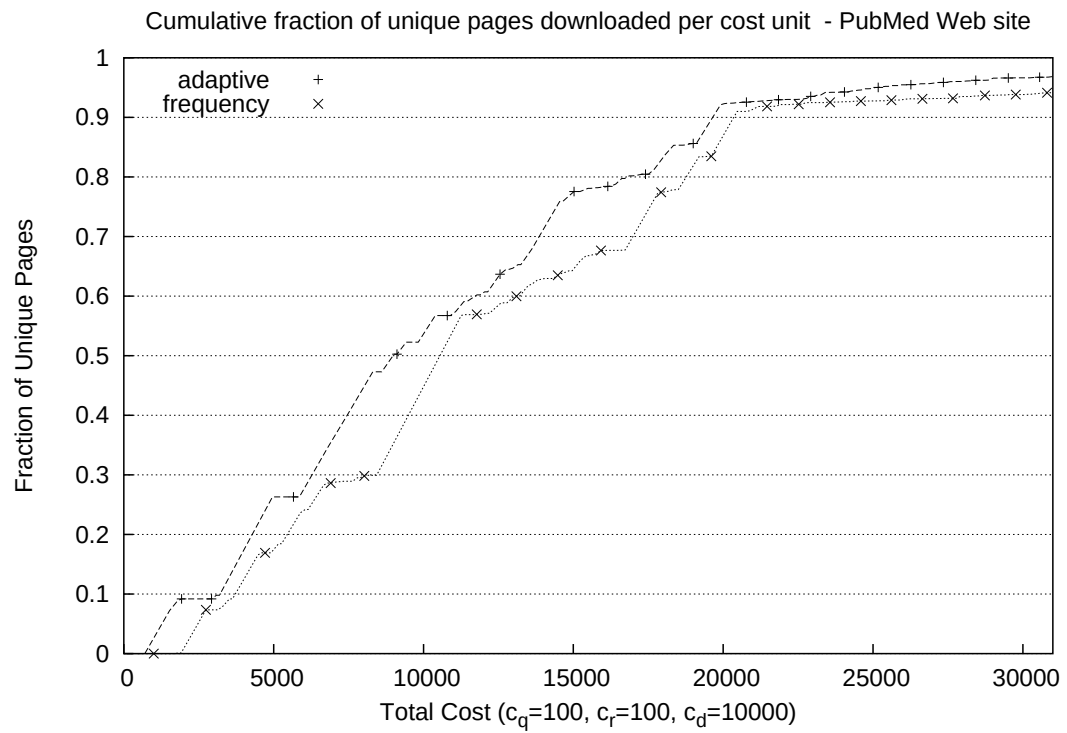


Figure 2.16: Coverage of PubMed after incorporating the document download cost

necessarily belong to the English dictionary. On the contrary, the generic-frequency algorithm is restricted by the language of the corpus that was analyzed before the crawl. In our case, the generic-frequency algorithm used a 5.5-million English word corpus, so as a result, it was limited to querying documents that could match only English words.

2.5 Related Work

In this section we review the literature related to the topics that we covered in the current chapter. The various approaches proposed by the research community for making the information on the Hidden Web widely accessible can be broadly divided based on two dimensions.

The first dimension is whether the Hidden-Web repositories are *cooperative* or *uncooperative*. If the Hidden-Web repositories are *cooperative* they allow direct access to a third-party application interested in downloading the Hidden-Web data. However, since this is typically done through the use of specialized protocols (e.g. Z39.50 [Moe92], SDLIP [PBL00]), the application needs to implement these protocols in order to communicate with the Hidden-Web site. In the case where a Hidden-Web site is *uncooperative*, we need to issue queries through the site's Web interface because there is no direct access to the Hidden-Web data.

The second dimension is whether our goal in making the Hidden-Web data widely accessible is to perform “on-the-fly” *data integration* or download the Hidden-Web data to a central *data warehouse* and make it accessible from this central location. When we perform *data integration*, the main idea is to direct a user's query to a number of sources and then aggregate the relevant results. Because *data integration* is typically performed in real time it generally leads to fresher results. However, because

	Cooperative sources	Uncooperative sources
Data integration	Z39.50 [Moe92], SDLIP [PBL00]	Gloss [GGT99],[IGS01], [IG02],[CS96], [LLC04], [HC03]
Data warehousing	OAI [LS01], ARC [LMZ01], DP9 [LMZ02]	[NZC04], [CCD99], [CC01], [BF04], [RG01], [CCH03], [IAJ06]

Table 2.2: Related work organized according to whether the Hidden-Web sources are *cooperative*, and whether we are interested in information *integration* or *warehousing*.

we are contacting the Hidden-Web sources every time there is a user query, we can put significant processing burden to the Hidden-Web sources. On the other hand, when we create a central *data warehouse*, we answer the user queries based on our own local copy of the data, but we need to periodically update our copy.

Overall, Table 2.2 shows the related work organized based on the two dimensions that we have just discussed. In the remainder of this section we go over the related work and compare it to the ideas that we covered in the current chapter. In our work, we mainly focused on enabling access to uncooperative Hidden-Web sources by downloading their data at a central location.

2.5.1 Cooperative data sources

2.5.1.1 Data integration

In order to make documents in multiple textual databases accessible from a central location, the research community has proposed a number of interoperability approaches, such as the Z39.50 protocol [Moe92] or the Simple Digital Library Interoperability Protocol (SDLIP) [PBL00].

Z39.50 [Moe92] provides an application-layer client/server protocol for the automated communication between computers while performing information search and retrieval tasks. Z39.50 defines a wide range of functions, such as a machine-independent representation of queries, a wide collection of attributes that can be used in queries, or capabilities for retrieving document fragments. SDLIP [PBL00] is an interoperability protocol similar to Z39.50, but is more lightweight and was created mainly for the exchange of information over the Web.

Both protocols, are geared towards *cooperative* data sources where we can perform *data integration* and are best suited for systems that manage bibliographic data. Library of Congress [Lib] implements Z39.50 today, and UC Berkeley provides SDLIP-based access to the University of California's library catalog [PBL00]. Our approach described in this chapter may not be directly applicable to the cases where the Hidden-Web sites provide sequential access to their whole collection of data (e.g. through the *Scan* operation of Z39.50). However, our ideas can still be used on top of an interoperability protocol for the cases where only query-based access is available, or the Hidden-Web sites impose a limit in the number of the returned results.

2.5.1.2 Data warehousing

For the cases where, instead of performing “on-the-fly” integration, we are interested in downloading the information from *cooperative* Hidden-Web sources at a central location, the research community has proposed metadata “harvesting” approaches, such as the Dublin Core [Dub], the Open Archives Initiative (OAI) [LS01], ARC [LMZ01] and DP9 [LMZ02].

OAI [LS01] is a generic framework that facilitates the collection of content from different Web sources. At a high level, it defines methods for harvesting metadata records from repositories by using HTTP and XML. The main idea [LMZ01] is to collect standardized metadata (e.g. based on the Dublin Core Metadata Initiative [Dub]), at a central location and then create useful services based on this centrally harvested data. DP9 [LMZ02] is a gateway service that aims at creating links to the documents within OAI-compliant repositories, so that typical crawlers can discover the data by following links.

Metadata harvesting approaches essentially assume that the Hidden-Web databases willingly share some of their metadata and/or documents to help a third-party search engine to index the documents. Our approach assumes uncooperative databases that do not share their data publicly and whose documents are accessible only through search interfaces. The metadata harvesting approaches can be of great benefit when indexing repositories from educational and bibliographic domains because they typically contain information of great quality. Unfortunately, metadata harvesting implementations (such as OAI [LS01]) are not very popular in the overall Web today mainly due to the fact that search engines tend to discard the metadata if they come from Web repositories that cannot be trusted.

2.5.2 Uncooperative data sources

2.5.2.1 Data integration

There exists a large body of work studying how to identify the most relevant databases given a user's query [GGT99, IGS01, IG02, CS96, LLC04, HC03], when the databases are *uncooperative*, providing only query-based access to their data. This body of work is often referred to as *metasearching* or *database selection* problem over the Hidden Web. The main idea is to direct a user's query to a multitude of *relevant* databases, retrieve the results, aggregate them and then present them to the users in a unified way.

For example, GLOSS [GGT99] uses *content summaries* (i.e. document frequency lists) and selects the databases that are more similar to the user's query. The method presented in [IGS01] exploits only the number of matches of the queries in order to identify the topical category of a database. [IG02] suggests the use of *focused probing* to classify databases into a topical category, so that, given a query, a relevant database can be selected based on its topical category.

[CS96] presents a system that can construct topic-specific hierarchies from text documents using a rule-based classifier, and then derives queries from the hierarchy in order to retrieve additional content. In [LLC04] the system contacts some databases adaptively, on-the-fly, in order to acquire a better view of which databases it should select for querying. In [HC03], the authors present a method for integrating the different schemas that the Hidden-Web sites employ.

One of the benefits of accessing the Hidden-Web through metasearching, is that the retrieved results are typically up-to-date because we are forwarding the user queries to the Hidden-Web sites and retrieve the results from the sources in real time. Overall, our vision is different from the metasearching body of work in that our goal is *download* and *index* the Hidden pages at a central location in advance, so that users can access

all the information at their convenience from one single location. In Chapter 4 we study effective methods of updating our central repository of Hidden-Web pages and maintain it up-to-date, so that we can also provide fresh information to the users.

2.5.2.2 Data warehousing

The idea of automatically issuing queries to a database and downloading the results has been previously used in different contexts.

For example, in [CCD99, CC01], Callan and Connel try to acquire an *accurate language model* by collecting a uniform random sample from the database. In [LG98] Lawrence and Giles issue random queries to a number of Web Search Engines in order to estimate the *fraction* of the Web that has been indexed by each of them.

In a similar fashion, Bharat and Broder [BB98] issue random queries to a set of Search Engines in order to estimate the *relative size and overlap* of their indexes. In [BF04], Barbosa and Freire experimentally evaluate methods for building multi-keyword queries that can return a large fraction of a document collection.

Our work differs from the previous studies in two ways. First, our work described in this chapter provides a theoretical framework for analyzing the process of generating queries for a database and examining the results, which can help us better understand the effectiveness of the methods presented in the previous work. Second, we apply our framework to the problem of Hidden-Web crawling and demonstrate the efficiency of our algorithms.

In [IAJ06], Ipeirotis et al. study Hidden-Web sites that are both *searchable* (i.e. they provide their content through a query-based interface) and *crawlable* (i.e. they also provide links to their content). They present algorithms for determining whether it is best to issue queries or follow links in order to retrieve a given portion (e.g. 70%) of

the Hidden-Web site. Our work is applicable to Hidden-Web sites that do not provide link-based access and our goal is to retrieve all of a given Hidden-Web site's pages.

In a recent study, Raghavan and Garcia-Molina [RG01] present an architectural model for a Hidden-Web crawler. The main focus of this work is to *learn* Hidden-Web query interfaces, not to generate queries automatically. The potential queries are either provided manually by users or collected from the query interfaces. In contrast, our main focus is to generate queries automatically without any human intervention.

Cope et al. [CCH03] propose a method to automatically detect whether a particular Web page contains a search form. This work is complementary to ours; once we detect search interfaces on the Web using the method in [CCH03], we may use our proposed algorithms to download pages automatically from those Web sites.

Reference [AIG03] reports methods to estimate what fraction of a text database can be eventually acquired by issuing queries to the database. In [AG03] the authors study query-based techniques that can extract *relational data* from large text databases. Again, these works study orthogonal issues and are complementary to our work.

2.6 Summary

Traditional crawlers normally follow links on the Web to discover and download pages. Therefore they cannot get to the Hidden-Web pages which are only accessible through query interfaces. In this chapter, we studied how we can build a Hidden-Web crawler that can automatically query a Hidden-Web site and download pages from it. We proposed three different query generation policies for the Hidden Web: a policy that picks queries at *random* from a list of keywords, a policy that picks queries based on their *frequency* in a generic text collection, and a policy which *adaptively* picks a good query based on the content of the pages downloaded from the Hidden-Web

site. Experimental evaluation on 4 real Hidden-Web sites shows that our policies have a great potential. In particular, in certain cases the *adaptive* policy can download more than 90% of a Hidden-Web site after issuing approximately 100 queries. Given these results, we believe that our work provides a potential mechanism to improve the search-engine coverage of the Web and the user experience of Web search.

CHAPTER 3

Evolution of Searchable Web Sites

3.1 Introduction

Given that Web pages are changing constantly, search engines need to update their index periodically in order to keep up with the evolving Web. An obsolete index leads to irrelevant or “broken” search results, wasting users’ time and causing frustration. In this dissertation, we study the evolution of a set of Web sites that provide query-based access to their pages. In this way we can get a better understanding on how our search engine should cope with the evolving pages on the Hidden Web.

A number of existing studies have already investigated the evolution of the Web [FMN03, BC00, MM00, CG00a, DFK97]. While some parts of our study have commonalities with the existing work, we believe that the following aspects make our study unique, revealing new and important details of the evolving Web.

- *Searchable Web sites*: Since we are interested in building a Hidden-Web search engine, we focused on Web databases that provide search-based access to their Web pages. Overall, our study and experiments involved 154 searchable Web sites, which we monitored weekly for a period of one year.
- *Link-structure evolution*: Search engines rely on both the content and the link structure of the Web to select the pages to return. For example, Google uses PageRank as their primary ranking metric, which exploits the Web link structure

to evaluate the importance of a page [BP98]. In this respect, the evolution of the link structure is an important aspect that search engines should know, but not much work has been done before. As far as we know, our work is the first study investigating the evolution of the link structure.

- *New pages on the Web*: While a large fraction of existing pages change over time, a significant fraction of “changes” on the Web are due to new pages that are created over time. In this dissertation, we study how many new pages are being created every week, how much new “content” is being introduced and what are the characteristics of the newly-created pages.
- *Search-centric change metric*: We study the changes in the existing Web pages using metrics directly relevant to search engines. Search engines typically use variations of TF.IDF distance metric to evaluate the relevance of a page to a query, and they often use an inverted index to speed up the relevance computation. In our dissertation, we measure the changes in the existing pages using both 1) the TF.IDF distance metric and 2) the number of new words introduced in each update. The study of the TF.IDF distance will shed light on how much “relevance” change a page goes through over time. The number of new words will tell us what fraction of an inverted index is subject to updating.

3.2 Experimental setup

To collect Web history data for our evolution study, we downloaded pages from 154 “popular” searchable Web sites (e.g., `acm.org`, `hp.com`, `oreilly.com`; see [Web] for a complete listing) every week from October 2002 until October 2003, for a total of 51 weeks. In this section, we explain how we selected the sites for our study and describe how we conducted the crawls of those sites. We also present a few general

statistics about the data we collected.

3.2.1 Selection of the sites

In selecting the sites to monitor, we wanted to pick a “representative” yet “interesting” sample of the searchable Web. By representative, we mean that our sample should span various parts of the Web, covering a multitude of topics.¹ By interesting, we mean that a reasonably large number of users should be interested in the sites, as search engines typically focus their resources on maintaining these sites the most up to date.

To obtain such a sample, we decided to pick roughly the five top-ranked pages from a subset of the topical categories of the Google Directory [gooa]. Google Directory reuses the data provided by the Open Directory Project [odp], and maintains a hierarchical listing of Web sites categorized by topic. Sites within each category are ordered by PageRank, enabling users to identify sites deemed to be of high importance easily. From the Web sites inside Google Directory, we picked only those sites that provided a search interface over their contents, thus considering the contents of those Web sites as “hidden” behind their respective search interfaces. By selecting sites from each topical category, we believe we made our sample “representative.” By picking only top-ranked sites, we believe we make our sample “interesting.” A complete list of sites included in our study can be acquired from [Web].

3.2.2 Download of pages

From the 154 Web sites we selected for our study, we downloaded pages every week over a period of almost one year. Our weekly downloads of the sites were thorough in

¹Our dataset constitutes a representative sample of the topical categories on the Web. As we will see later in Section 3.3.2.1, the trends that we observed from our dataset still hold for a completely random sample of the Web.

all but a few cases: starting from the root pages of the Web sites, we downloaded in a breadth-first order either *all* reachable pages in each site, or all pages until we reached a maximum limit of 200,000 pages per site. Since only four Web sites (out of 154) contained more than 200,000 pages,² we have captured a relatively complete weekly history of these sites. Capturing nearly complete snapshots every week is important for our purposes, as one of our main goals is to study the creation of new pages.

The total number of pages that we downloaded every week ranges from 3 to 5 million pages, with an average of 4.4 million pages. The size of each weekly snapshot was around 65 GB before compression. Thus, we currently have a total of 3.3 TB of Web history data, with an additional 4 TB of derived data (such as links, shingles, etc.) used for our various analyses. When we compress the weekly snapshots using the standard zlib library, the space footprint is reduced to about one third of the original.

Table 3.1 reports the fraction of pages included in our study that belong to each high-level domain. The `misc` category contains other domains including regional ones such as `.uk`, `.dk`, `.jp` etc. The distribution of domains for pages in our study roughly matches the general distribution of domains found on the Web [BBC00].

3.3 What's new on the Web?

In this section, we focus on measuring what is new on the Web each week. In particular, we attempt to answer questions such as: How many new pages are created every week? How much new content is created? How many new links? We begin by studying the weekly birth rate of pages.

For our analysis in Sections 3.3.1 and 3.3.2 we treat each unique URL as a distinct unit. Then, in Section 3.3.3 we measure the shift in the collective content of all pages,

²The sites containing more than 200,000 pages were `www.eonline.com`, `www.hti.umich.edu`, `www.pbs.org` and `www.intelihealth.com`.

Domain	Fraction of pages in domain
.com	41%
.gov	18.7%
.edu	16.5%
.org	15.7%
.net	4.1%
.mil	2.9%
misc	1.1%

Table 3.1: Distribution of domains in our crawls.

to filter out the effect of content duplication among URLs.

3.3.1 Weekly birth rate of pages

We first examine how many new pages are created every week. That is, for every snapshot, we measure the fraction of the pages in the snapshot that have not been downloaded before and we plot this number over time. This fraction represents the “weekly birth rate” of Web pages. We use the URL of a page as its identity, and consider a page “new” if we did not download any page with the same URL before. Under this definition, if a page simply changes its location from URL A to URL B, we consider that a new page B has been created. (Later in Section 3.3.3 we measure how much new “content” is introduced every week, which factors out this effect.)

In Figure 3.1 we show the weekly birth rate of pages, with week along the horizontal axis. The line in the middle of the graph gives the average of all the values, representing the “average weekly birth rate” of the pages. From the graph we can observe that the average weekly birth rate is about 8%. That is, 8% of pages downloaded by an average weekly crawl had not been downloaded by any previous crawl. Scaling

up from our data (which, by design, is biased toward popular pages), and assuming the entire Web consists of roughly 11.5 billion pages [GS05], we conjecture that there may be around 920 million new pages created every week (including copies of existing pages and relocated pages). Admittedly, this number may not be fully accurate because our study focuses on popular pages. However, it does give us a ball-park figure.

We also observe that approximately once every month, the number of new pages being introduced is significantly higher than in previous weeks. For example, the bars are higher in weeks 7, 11, 14, etc. than their previous weeks. Most of the weeks with the higher birth rate fall close to the end of a calendar month. This fact implies that many Web sites use the end of a calendar month to introduce new pages. Manual examination of the new pages in these “high birth rate” weeks revealed that a number of such pages contain job advertisements or portals leading to archived pages in a site. For the most part, however, we could not detect any specific pattern or topical category for these pages.

3.3.2 Birth, death, and replacement

In our next experiment, we study how many new pages are created and how many disappear over time.³ We also measure what fraction of pages on our Web sites is replaced with new pages after a certain period. For these purposes, we compare our weekly snapshots of the pages against the first snapshot and measure 1) how many pages in the first snapshot still remain in the n th-week snapshot, and 2) how many pages in the n th week snapshot do not exist in the first snapshot. For all the comparisons presented here, the URLs of the crawled pages were canonicalized.

Figure 3.2 shows the result. The horizontal axis of this graph plots the week and the

³We assume that a page disappeared if our crawler received an HTTP 404 response for that particular page, or we could not download the page (due to timeouts, etc.) after three attempts.

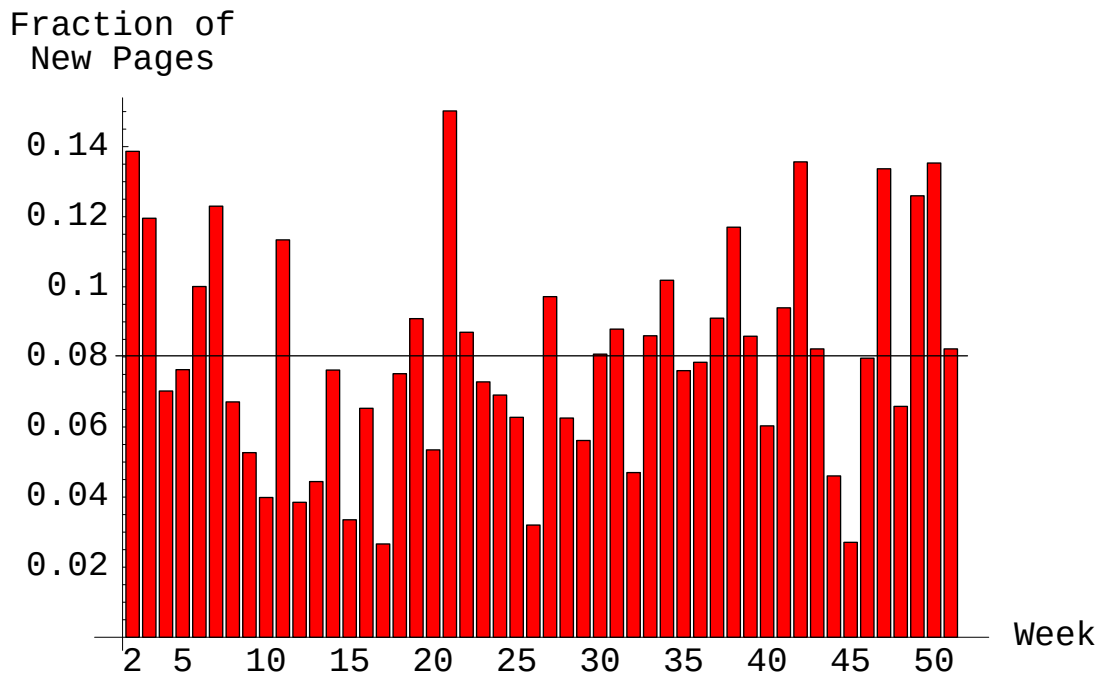


Figure 3.1: Fraction of new pages between successive snapshots.

vertical axis shows the number of pages that we crawled in the given week. The bars are normalized such that the number of pages in the first week is one. (We downloaded 4.8 million pages in the first week.) The dark bars represent the number of first-week pages that were still available in the given week. The light bars represent the number of pages that were created since the first week (i.e., the pages that exist in the given week but did not exist in the first week). For example, the size of the second-week snapshot was about 80% of that of the first week, and we downloaded about 70% of the first-week pages in the second week.

The observable fluctuations in our weekly crawl sizes (most noticeable for week 45) are primarily due to technical glitches that are difficult to avoid completely. While collecting our data, to minimize the load on the Web sites and our local network, we ran our crawler in a slow mode. It took almost a full week for the crawler to finish each crawl. During this time, a Web site may have been temporarily unavailable or our

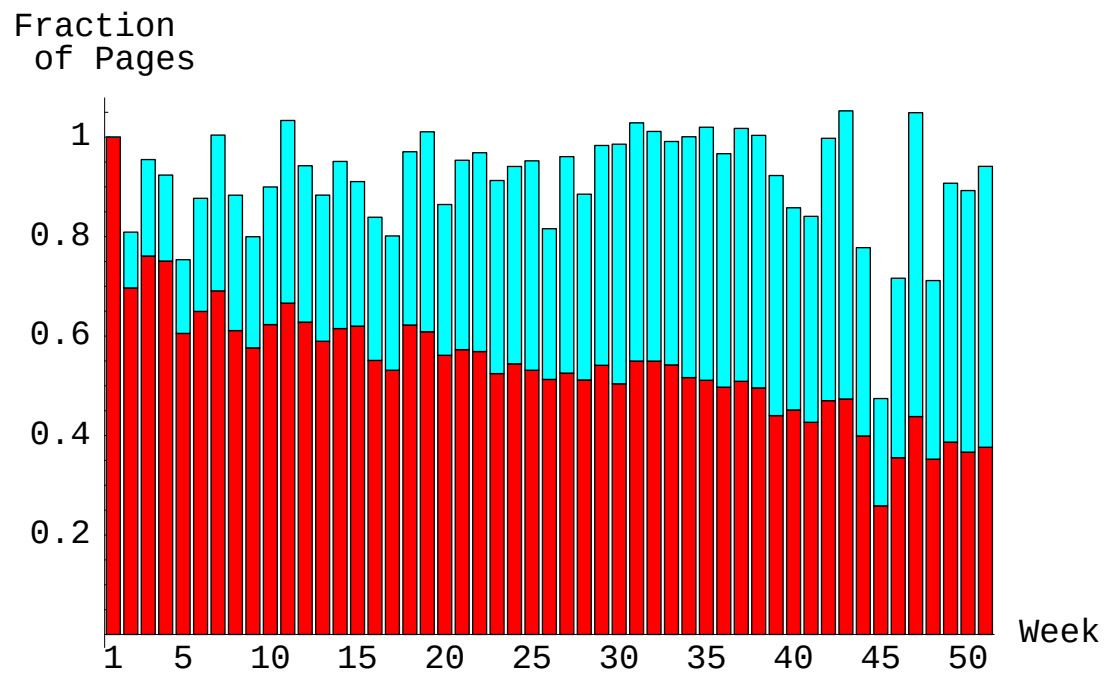


Figure 3.2: Fraction of pages from the first crawl still existing after n weeks (dark bars) and new pages (light bars).

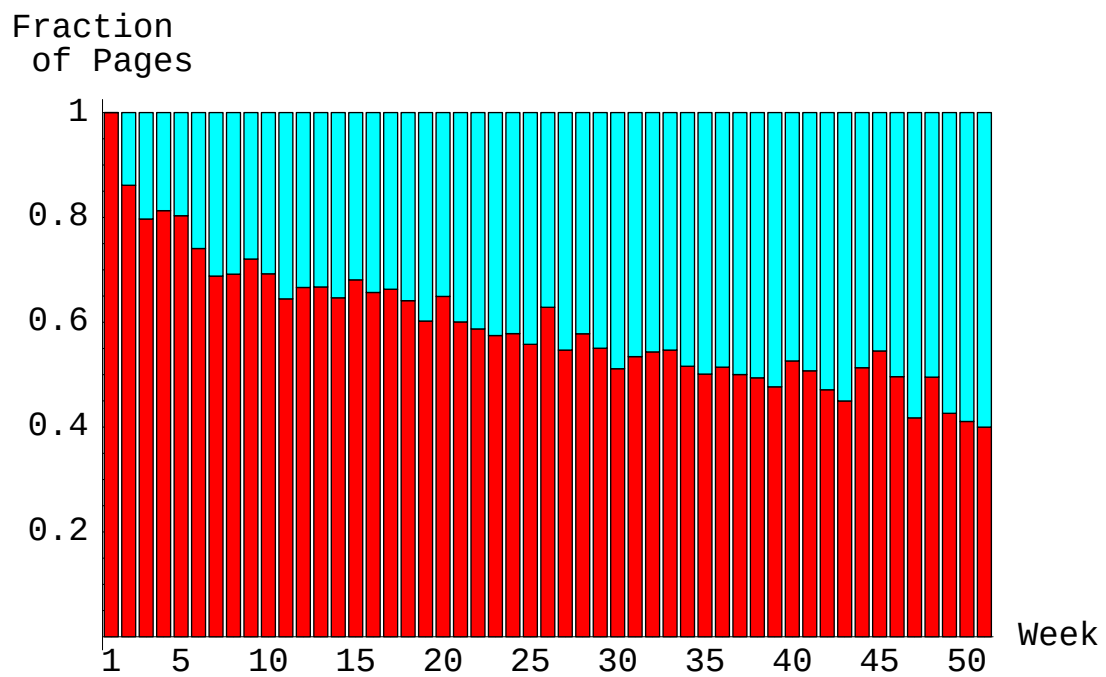


Figure 3.3: Normalized fraction of pages from the first crawl still existing after n weeks (dark bars) and new pages (light bars).

local network connection may have been unreliable. To be robust against short-lived unavailabilities our crawler makes up to three attempts to download each page. Still, in certain cases unavailabilities were long-lived and our crawler was forced to give up. Since these glitches were relatively minor in most cases (except in the 45th week when one of our crawling machines crashed), we believe that our results are not significantly affected by them.

By inspecting the weeks with the highest bars in Figure 3.2 and taking glitches with our crawling into account, we find that the total number of pages available from the 154 sites in our study remained more or less the same over the entire 51-week period of our study. However, they are not all the same pages. Instead, existing pages were replaced by new pages at a rapid rate. For example, after one month of crawling (week 4), only 75% of the first-week pages were still available (dark portion of the graph at week 4), and after 6 months of crawling (week 25), about 52% are available.

A normalized version of our graph is shown in Figure 3.3, with the numbers for each week normalized to one to allow us to study trends in the fraction of new and old pages. After six months (week 25), roughly 40% of the pages downloaded by our crawler were new (light bars) and around 60% were pages that also occurred in our first crawl (dark bars). Finally, after almost a year (week 51) nearly 60% of the pages were new and only slightly more than 40% from the initial set was still available. It took about nine months (week 39) for half of the pages to be replaced by new ones (i.e., half life of 9 months).

To determine whether the deletion rate of pages shown in Figure 3.3 follows a simple trend, we used linear regression to attempt to fit our data using linear, exponential, and inverse-polynomial functions. The deletion rate did not fit any of these trends well. The best match was with a linear trend, but the R-squared value was still very low at 0.8.

3.3.2.1 Generalizing to the entire Web

Our results can be combined with results from recent study by the Online Computer Library Center (OCLC) [ocl] to get a picture of the rate of change of the entire Web. The OCLC collects an annual sample of the Web and studies various trends pertaining to the nature of Web sites. One of the experiments that the OCLC has conducted over the last few years is to estimate how the number of available Web sites changes over time. From years 1998 to 2002, OCLC has performed systematic polling of IP addresses to estimate the total number of available Web sites. They have also measured what fraction of Web sites are still available after k years.

The result of this OCLC study is shown on Figure 3.4. In the figure, the horizontal axis represents the year of measurement. The overall height of each bar shows the total number of Web sites available at the given year, relative to the number of sites available in 1998. In 1998 the number of the publicly-accessible Web sites was estimated to be 1.4 million. The dark bottom portion of the bar represents the fraction of the Web sites that existed in 1998 and were still available in the given year. The light portion represents the fraction of new Web sites that became available after 1998. From the graph, we can see that about 50% of Web sites go offline every year. For example, in 1999, half of the 1998 Web sites were still accessible.

Combining this result with ours, we may get an idea of how many pages on the entire Web will still be available after a certain period of time. The OCLC study shows that about 50% of existing Web sites remain available after one year. Our study shows that roughly 40% of the pages in each Web site remain available after one year. Therefore, we can speculate that only about $50\% \times 40\% = 20\%$ of today's Web pages will be accessible after one year.

Given this low rate of “survival” of Web pages, historical archiving as performed by, e.g., the Internet Archive [int], is of critical importance for enabling long-term

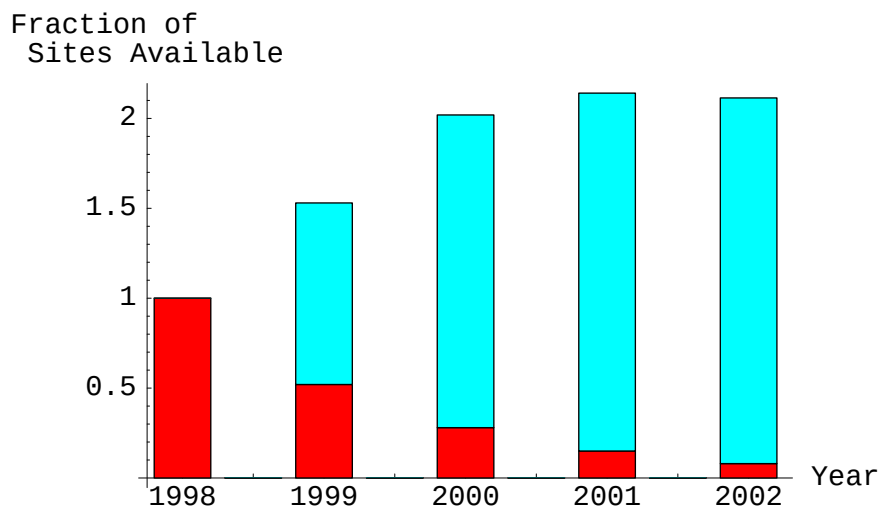


Figure 3.4: Percent of IP addresses identifying a Web site in Year A also identifying a Web site in Year B. For example, 56% of IP addresses identifying a Web site in the 1998 sample also identified one in 1999 sample. Taken from <http://wcp.oclc.org/>.

access to historical Web content. A significant fraction of pages accessible today are unlikely to be available after one year. Another implication of our finding applies to standard search engines that do not focus on access to historical content. Since search engine users tend to be very intolerant of broken links in search results, it is very important for search engines to keep abreast of page deletions and omit deleted pages from search results (or, alternatively, point to “cached” copies as Google [goob] and other search engines sometimes do, which effectively extends the lifetime of pages).

Finally, from Figure 3.4, we observe that growth in the number of the Web sites has slowed significantly in recent years. While the number of available Web sites increased by 50% between year 1998 and 1999, the total number of available sites did not change much since year 2000. Given that the number of pages within popular sites does not appear to grow significantly over time (our finding discussed above), there are two remaining potential sources of growth on the Web.

First, it may be the case that relatively unpopular sites are growing. Although our study does not focus on the behavior of unpopular sites, as a small side project we did measure growth for a small random sample of 100 sites on the Web over a period of two months. Our findings for those sites matched those for popular sites: the overall number of pages remained nearly constant over time. Further measurements over a larger scale are needed to verify this preliminary and as yet inconclusive finding.

The second potential source of growth on the Web may stem from increase in the size of pages. While the total number of pages may be leveling off, perhaps it is the case that pages are growing larger over time. In Figure 3.5 we plot the average page size of each of our snapshots. The horizontal axis plots the week and the vertical axis shows the average page size in a given week, normalized so that average size in the first week equals one. While we see wide fluctuations, a clear upward trend exists in the graph, although it has very mild slope. Given these results, we suspect that the

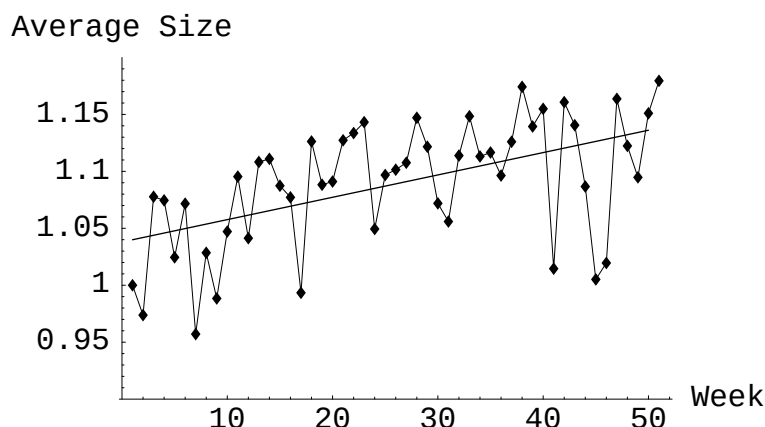


Figure 3.5: Average page sizes in our snapshots over time.

current growth of the Web is mainly driven by the increase in the size of pages over time.

3.3.3 The creation of new content

While we measured the rates of creation and deletion of Web pages in previous sections, we did not address how much “new content” is being introduced over time. That is, even when a new page is created, the page may be a mere copy of an existing page in which case it does not contribute any new content to the Web. To quantify the amount of new content being introduced, we use the shingling technique as described in [BDG95, BGM97]. From every page we exclude the HTML markup and we view the page as an ordered sequence of lower-cased words. A w -shingle is a contiguous ordered subsequence of w words. That is, we group w adjacent words of the page to form a w -shingle, possibly wrapping at the end of the page, so that all words in the page start a shingle.

To measure how much content is created we computed the shingles for all pages included in our study and compared how many new unique shingles are introduced

over time. We wanted to answer the following questions: Out of all unique shingles that existed in the first week, how many of them still exist in the n th week? How many unique shingles in the n th week did not exist in the first week? By measuring the number of unique existing and newly appearing shingles, we can learn how much “new content” is being introduced every week. For the experiments presented in this section we used a shingle size of $w = 50$, which roughly corresponds to the number of words in a typical paragraph.

The result of our shingle measurements is shown in Figure 3.6. The horizontal axis plots time in weeks and the vertical axis shows the total number of unique shingles present each week, relative to the first week. The first week has approximately 4.3 billion unique shingles. The darkly colored, lower portion of each bar shows the number of first-week shingles available in the n th week. The lightly colored, upper portion shows the number of new shingles that did not exist in the first week. To factor out fluctuation in the total number of shingles and focus on the trends in relative terms, we show a normalized version of the graph in Figure 3.7, where the total number of unique shingles in each weekly crawl is normalized to one.

By comparing Figure 3.7 with Figure 3.3, we can see that new shingles are created at a slower rate than new pages. It takes nine months for 50% of the pages to be replaced with new ones, but even after nearly one year, more than 50% of the shingles are still available. On average, each week around 5% of the unique shingles were new, i.e., not present in any previous week. It is interesting to contrast this figure with our finding from Section 3.3.1 that, on average, roughly 8% of pages each week were new (when identified solely based on URLs). By combining the two results, we determine that at most $5\%/8\%=62\%$ of the content of new URLs introduced each week is actually new.

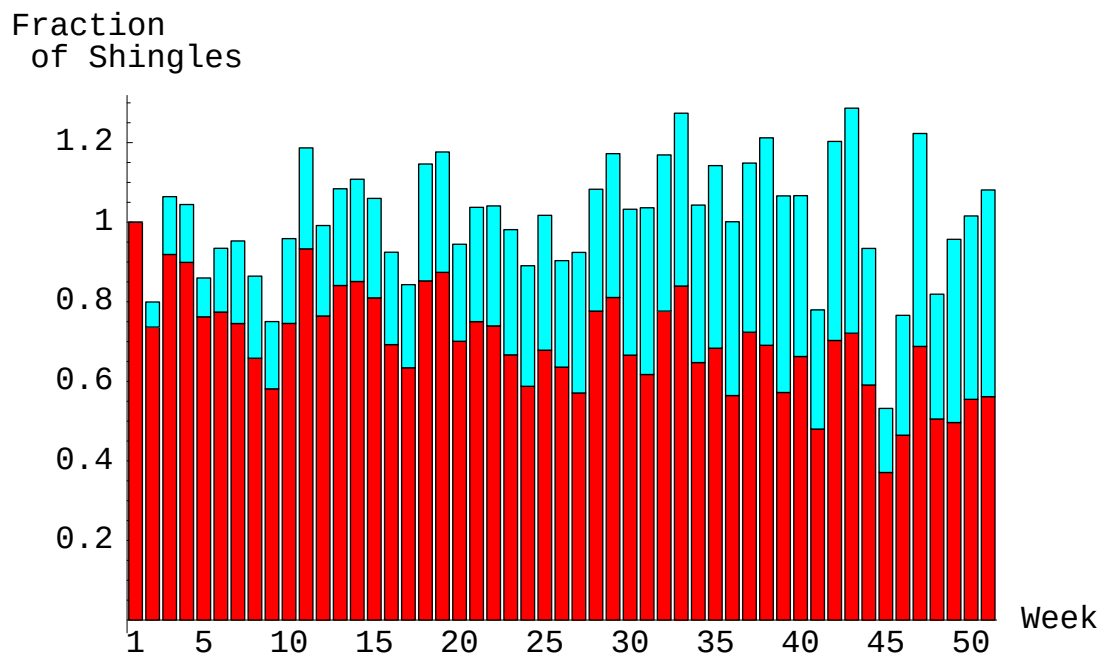


Figure 3.6: Fraction of shingles from the first crawl still existing after n weeks (dark portion of bars) and shingles newly created (light portion).

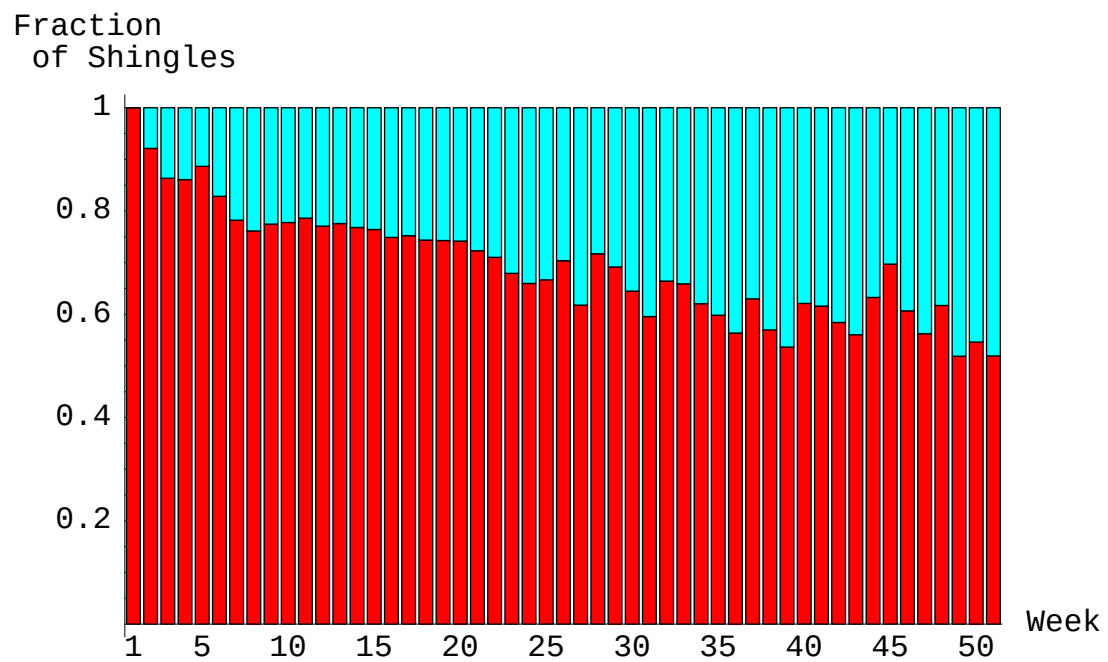


Figure 3.7: Normalized fraction of shingles from the first crawl still existing after n weeks (dark portion of bars) and shingles newly created (light portion).

Fraction of Links

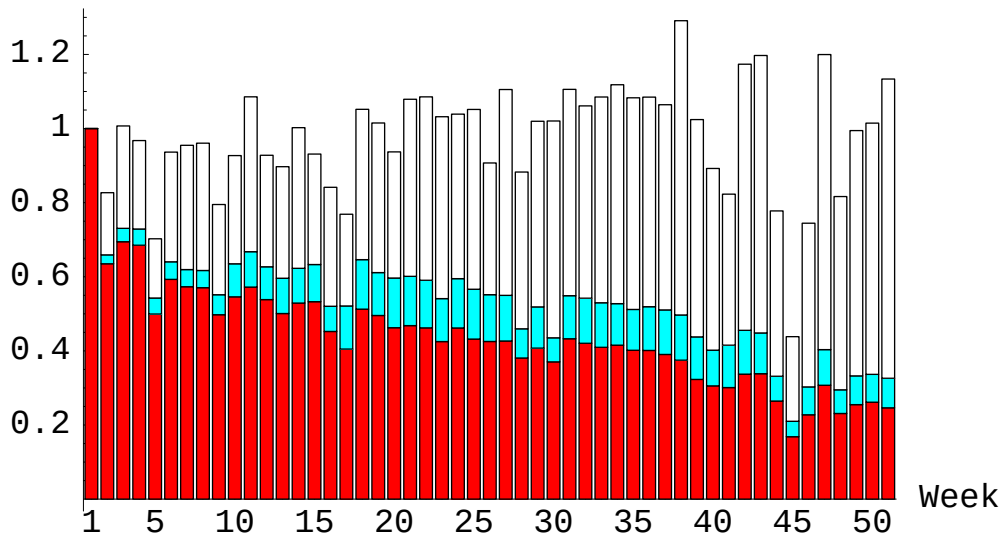


Figure 3.8: Fraction of links from the first weekly snapshot still existing after n weeks (dark/bottom portion of the bars), new links from existing pages (grey/middle) and new links from new pages (white/top).

3.3.4 Link-structure evolution

The success of Google has demonstrated the usefulness of the Web link structure in measuring the importance of Web pages. Roughly, Google's PageRank algorithm estimates the importance of a page by analyzing how many other pages point to the page. In order to keep up with the changing importance and popularity of Web pages, it is thus important for search engines to capture the Web link structure accurately. In this section we study how much the overall link structure changes over time. For this study, we extracted all the links from every snapshot and measured how many of the links from the first snapshot existed in the subsequent snapshots and how many of them are newly created.

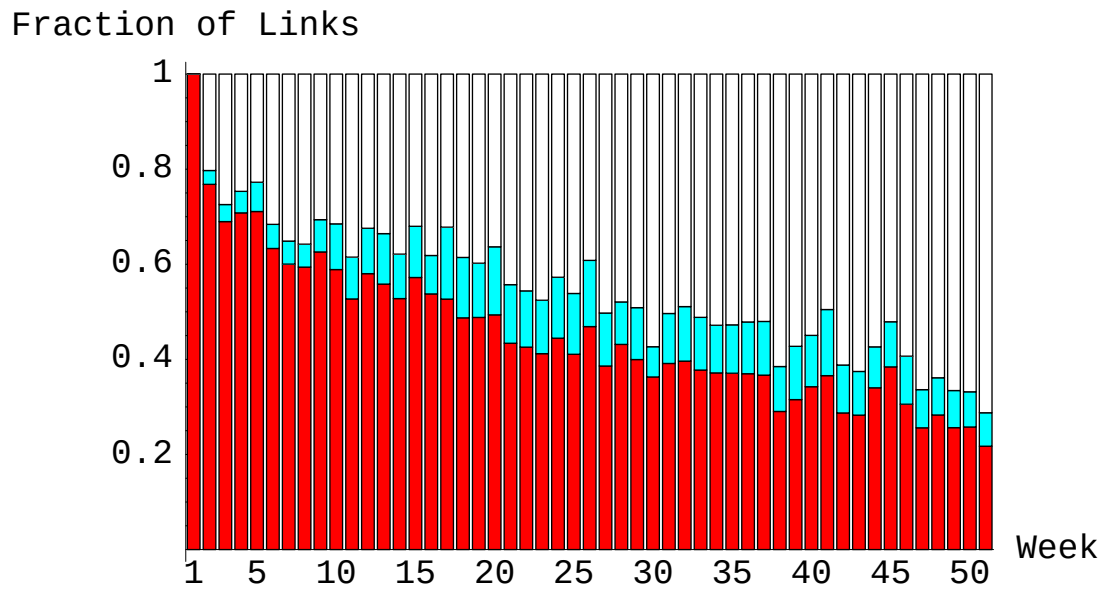


Figure 3.9: Normalized fraction of links from the first weekly snapshot still existing after n weeks (dark/bottom portion of the bars), new links from existing pages (grey/middle) and new links from new pages (white/top).

The result of this experiment is shown in Figure 3.8. The horizontal axis shows the week and the vertical axis shows the number of links in the given week. The height of every bar shows the total number of links in each snapshot relative to the first week. The dark-bottom portion shows the number of first-week links that are still present in the given week. The grey and white portions represent the links that did not exist in the first week: The grey portion corresponds to the new links coming from the “old” pages (the pages that existed in the first week), while the white portion corresponds to the new links coming from the “new” pages (the pages that did not exist in the first week). Figure 3.9 is the normalized graph where the total number of links in every snapshot is one.

From the figure, we can see that the link structure of the Web is significantly more dynamic than the pages and the content. After one year, only 24% of the initial links are available. On average, we measure that 25% new links are created every week, which is significantly larger than 8% new pages and 5% new content. This result indicates that search engines may need to update link-based ranking metrics (such as PageRank) very often. For example, given the 25% new links every week, a week-old ranking may not reflect the current ranking of the pages very well.

3.4 Changes in the existing pages

The previous experiment demonstrated that every week numerous pages disappear from our weekly snapshots and another set of pages is created. The pages that appear repeatedly in our weekly snapshots, however, do not all remain static. In this section we study the way in which the content of pages captured repeatedly by our weekly snapshots changes over time.

3.4.1 Change frequency distribution

In our first experiment, we investigate how often Web pages change on average. We begin by using the simplest definition of a change: we consider *any* alteration to a page as constituting a change. Later, we will consider a more refined notion of change (Section 3.4.2).

For this experiment we conducted a scan of our weekly snapshots to determine, for each page that appeared in all snapshots, the average interval between successive changes. For example if a particular page changed twice during our 51-week measurement period, its average change interval is $51/2=25.5$ weeks. We then grouped pages by change interval and obtained the distribution shown in Figure 3.10. Average change interval is plotted on the horizontal axis. The vertical axis shows the fraction of pages having each average change interval. Pages that did not change at all during our 51-week measurement period are counted in the bar on the far right, marked “inf.” The large gaps between bars toward the right side of Figure 3.10 correspond to average change intervals that cannot arise in a 51-week experiment.

From Figure 3.10 we observe that a significant fraction of pages (around 50%) that occurred in each weekly snapshot remained unchanged throughout the course of our study. Another quite large portion of pages changed very often: approximately 15% of pages underwent at least one change between each weekly download. These two extremes account for more than 65% of the pages. The remaining pages occurring in all snapshots have average change intervals ranging across the spectrum in a roughly U-shaped pattern, with most pages concentrated near one of the two extremes. The tendency is that most pages either change very frequently or very infrequently.

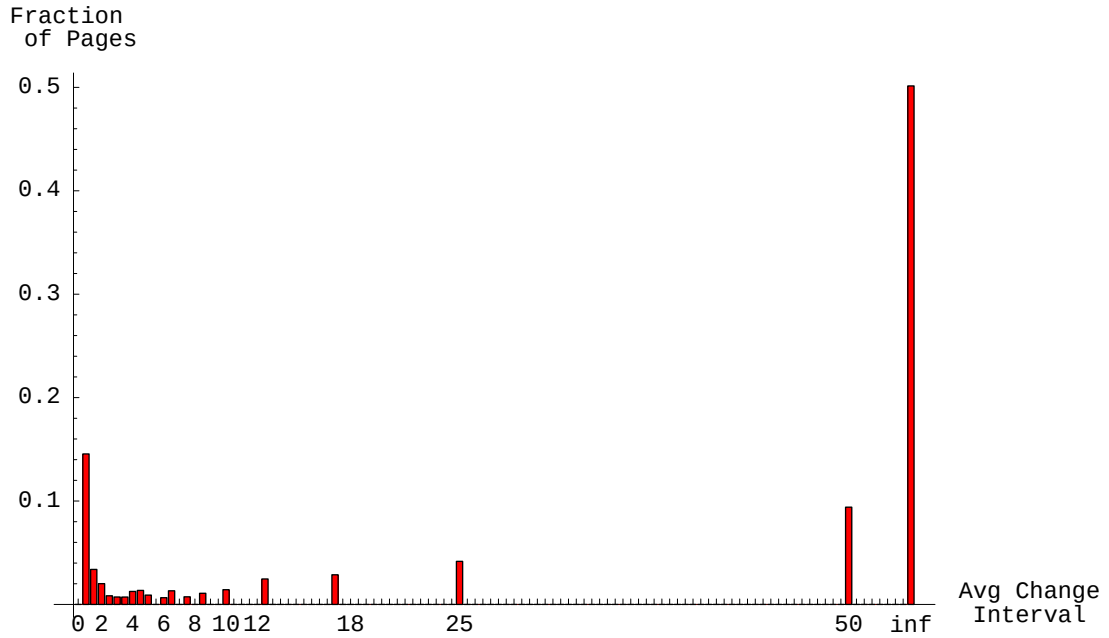


Figure 3.10: Distribution of the average change intervals of the pages.

3.4.2 Degree of change

In our previous experiment we used a very simple definition of change that only captures the *presence* of change, ignoring whether changes are major or minor ones. To delve deeper into the nature of changes undergone by Web pages over time, we now report on experiments designed to measure *degree* of change.

From the point of view of a Web search engine, degree of change is as important as, if not more important than, presence of change. Due to the immense scale and highly dynamic nature of the Web, search engines are faced with a constrained optimization problem: maximize the accuracy of the local search repository and index, given a constrained amount of resources available for (re)downloading pages from the Web and incorporating them into the search index. Search engines that ignore degree of change may waste precious resources downloading pages that have changed in only

trivial ways and have little impact on the quality of the search service. Effective search engine crawlers ignore insignificant changes and devote resources to incorporating important changes instead.

Our goal with these experiments was to get a handle on how degree of change may influence the design of highly effective search engine crawlers. Hence, we measured the distribution of degree of change using two metrics that are of relevance to typical search engines:

1. **TF.IDF Cosine Distance** Given two versions of a page p , say p_1 and p_2 , we calculate the TF.IDF cosine distance [SM83] between p_1 and p_2 . More precisely, supposing v_1 and v_2 are the TF.IDF weighted vector representations of p_1 and p_2 (excluding any HTML markup), we compute cosine distance as follows:

$$D_{cos}(p_1, p_2) = 1 - \frac{v_1 \cdot v_2}{\|v_1\|_2 \|v_2\|_2}$$

where $v_1 \cdot v_2$ is the inner product of v_1, v_2 and $\|v_i\|_2$ is the second norm, or length, of vector v_i .

2. **Word Distance** Given two versions of a page p , p_1 and p_2 , we measure how many words of text in p 's content have changed (we exclude any HTML markup). The word distance between p_1 and p_2 is defined as:

$$D_{word}(p_1, p_2) = 1 - \frac{2 \cdot |\text{common words}|}{|\text{words in } p_1| + |\text{words in } p_2|}$$

Note that both degree of change metrics are normalized so that all values are between zero and one, with 0 corresponding to no change and 1 indicating that the two versions differ completely.

The TF.IDF cosine distance metric (in its various forms) is the most commonly used method of determining relevance of documents to search queries based on content. Search engines typically rank search results using a combination of cosine distance and other factors (including link-based importance measures as discussed in Section 3.3.4). A small cosine distance change for a page generally translates to a relatively minor effect on result ranking for most search queries.

Word distance is also important from the perspective of search engine design. Word distance reflects the amount of work required to bring the search index up to date, assuming modifications are made incrementally to allow immediate searchability as in [LWP03]. Both metrics ignore the order in which terms appear on a page, i.e. they treat pages as “bags of words.” Doing so is consistent with the way in which typical search engines treat documents (with the exception of phrase matching). In contrast, the shingles metric (which we used in Section 3.3.3; it is also used by [FMN03]) is highly sensitive to the exact order of terms.

The distribution of TF.IDF cosine distance for all changes is shown in Figure 3.11. To ensure proper comparability across multiple weeks whose snapshots contained different numbers of pages, we selected a representative week (week 21) from which to obtain IDF weights to use in all of our TF.IDF calculations. The horizontal axis of Figure 3.11 shows cosine distance and the vertical axis shows the fraction of changes corresponding to the given distance. The dark bars show the distribution of cosine distances; the light bars give the cumulative distribution.

By examining Figure 3.11 we can see that most changes are very small, and concentrated on the far left portion of the graph. More than 80% of all changes resulted in a new version whose cosine distance was less than 0.2 from the old version. In fact, 65% of the changes had a cosine distance of less than 0.05. In light of this finding, we conclude that over half of the total changes recorded by our repeated crawls were

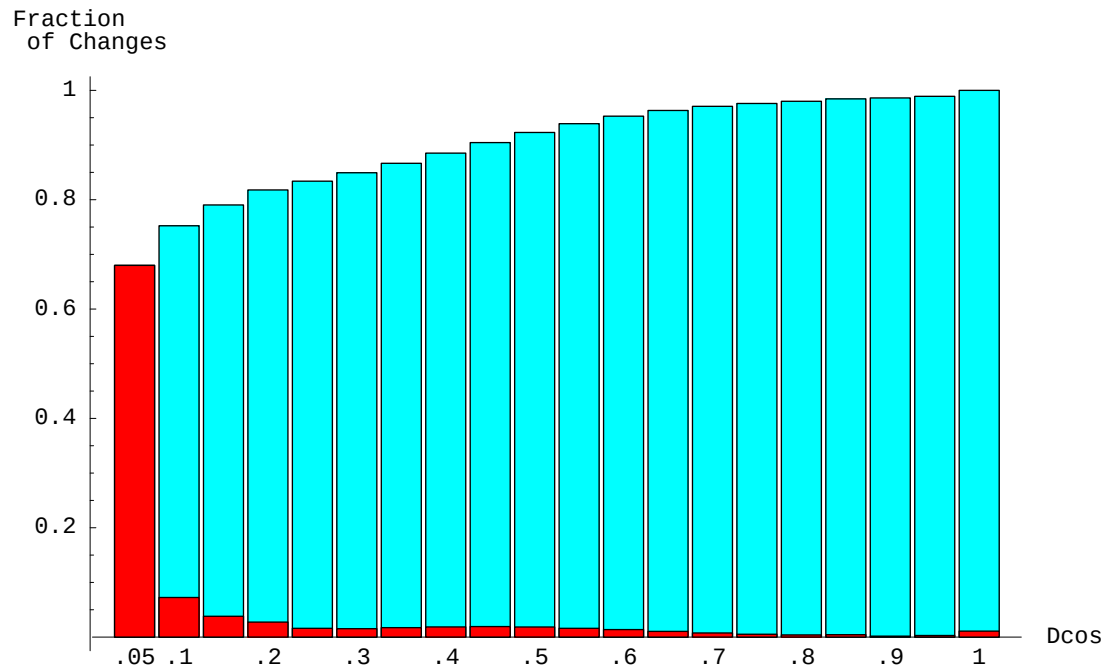


Figure 3.11: Distribution of cosine distance for all changes. Each dark bar corresponds to changes with cosine distance between the respective x-axis value and the previous one. For example, bin 0.1 corresponds to changes with cosine distance between 0.05 and 0.1. The light bars show the cumulative distribution.

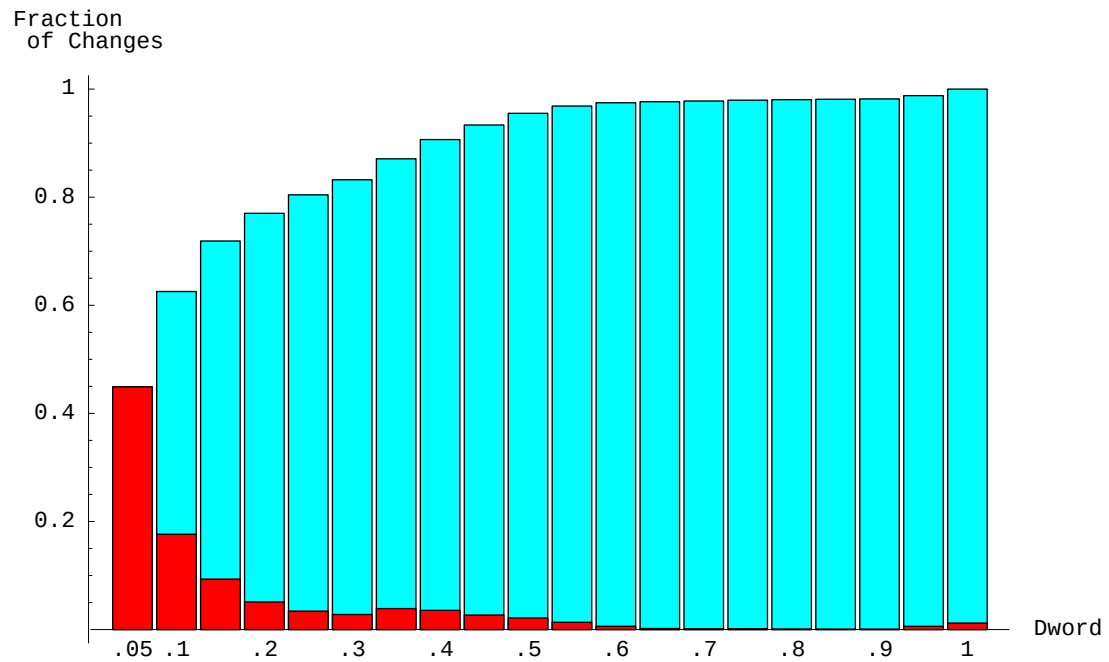


Figure 3.12: Distribution of word distance values for all changes. Each dark bar corresponds to changes with word distance between the respective x-axis value and the previous one. For example, bin 0.1 corresponds to changes with word distance between 0.05 and 0.1. The light bars show the cumulative distribution.

induced by operations that altered the content of a page very slightly. Such operations might be modifications to advertising material, counters, “last updated” tags, etc. We provide evidence to support this conjecture in Section 3.4.3.

Our finding coincides with that of [FMN03], which measured degree of change by counting the number of “shingles” affected by a change occurrence. However, we feel that the metric we have selected, TF.IDF cosine distance, may be more directly relevant to search engine crawler design. The observation that most Web page modifications influence cosine distance (and thus ranking accuracy for most queries) very little may have important implications. In particular, in light of this fact it is crucial that search engine crawlers facing resource limitations consider the degree of change, not just the presence of change, when choosing among pages to download and incorporate into the search index. Of course, fine-grained degree of change information can only be leveraged in a traditional search engine context if it is amenable to prediction. We study predictability of degree of change later in Section 3.5.

We now turn to measurements of word distance. Figure 3.12 shows the distribution of word distances for all changes detected. We again see that the vast majority of changes are relatively minor ones, although this phenomenon is less pronounced for word distances than it is for cosine distances. The difference between Figures 3.11 and 3.12 indicates that a moderate fraction of changes induce a nontrivial word distance (such as 0.2) while having almost no impact on cosine distance. These are changes that primarily affect words with high document frequency that are typically given low weight in search result ranking functions. Incorporating such changes into a search index incurs moderate overhead while adding relatively little benefit in terms of search result quality. This phenomenon and its implications for search engine design merit further study.

3.4.3 Degree and frequency of change

Our findings in Section 3.4.2 indicate that most changes are relatively minor ones, so it is important for search engine crawlers to take degree of change into account. We now investigate whether there is any correlation between frequency of change and degree of change. If so, then perhaps degree of change can be estimated indirectly by measuring frequency of change. For example, perhaps it is the case that pages that are modified very often (say, once every day) usually experience only a minor degree of change with each modification (e.g., swapping advertisements). Conversely, perhaps pages that are only modified occasionally (say, twice per year) undergo a large-scale overhaul with each modification.

In our next experiment we aimed to determine whether such a correlation exists. To check for correlation we grouped pages based on their average frequency of change (based on the “all-or-nothing” notion of change) and computed the average degree of change for changes in each group. Degree of change was measured using both the TF.IDF cosine distance and word distance metrics described in Section 3.4.2.

The result is shown in Figure 3.13. The horizontal axis represents the number of times (1 to 50) a page changed over the course of our 51 downloads. The vertical axis shows the average degree of change, for each change undergone by pages in each category. The line for D_{cos} corresponds to the cosine distance metric; the line for D_{word} corresponds to word distance. Under both metrics, the highest average degree of change per change occurrence is experienced by pages that either change very frequently (far right of the graph) or change very rarely (left side of the graph). This fact implies that the content of the pages that change very frequently (at least once per week) is significantly altered with each change. The same is true for the pages that change infrequently. Otherwise, no discernible trend is apparent.

To study the relationship between degree and frequency of change in more depth,

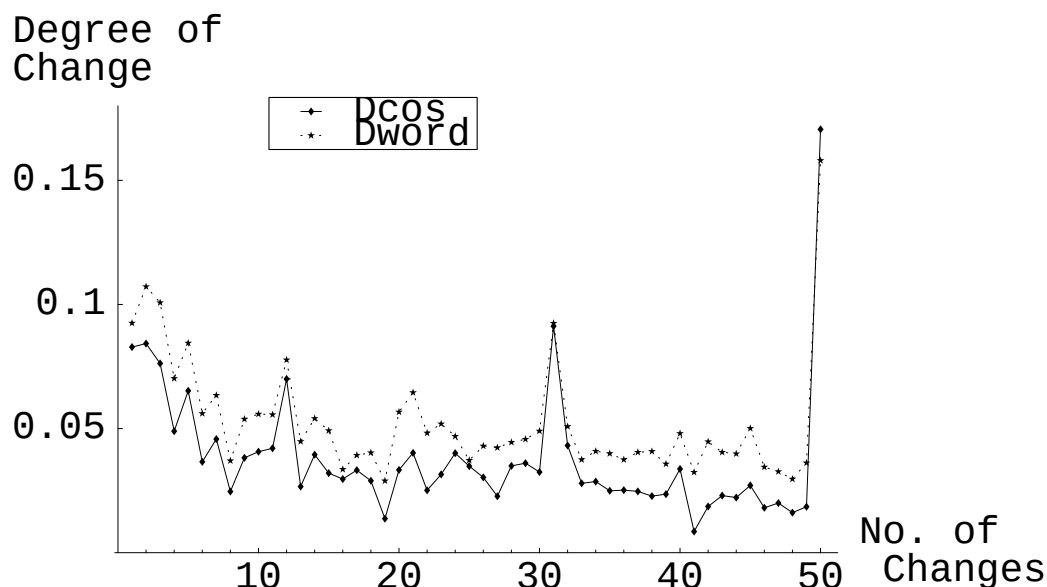


Figure 3.13: Relationship between degree of change and frequency of change.

we also measured *cumulative* degree of change grouped by change frequency: Figure 3.14 plots overall degree of change between the first and last version of each page, averaged across all pages within each change frequency group (horizontal axis).

By comparing Figures 3.13 and 3.14 we can see that for pages that changed at least weekly (i.e. 50 times during our 51-week measurement period), although each week a roughly 17% degree of change was measured on average (Figure 3.13), the cumulative degree of change after 50 weeks was only around 30% under both metrics (Figure 3.14). This finding suggests that the vast majority of modifications to these frequently updated pages tend to occur in the same portion(s) of the page. We inspected a small sample of such pages by hand, and found that in many cases repeated modification of a restricted portion of the content corresponds to aspects such as: weather, stock market, “news of the day” reports, counters, advertisement material containing small text strings (recall that our measurements factor out html tags, images, etc.), and “last

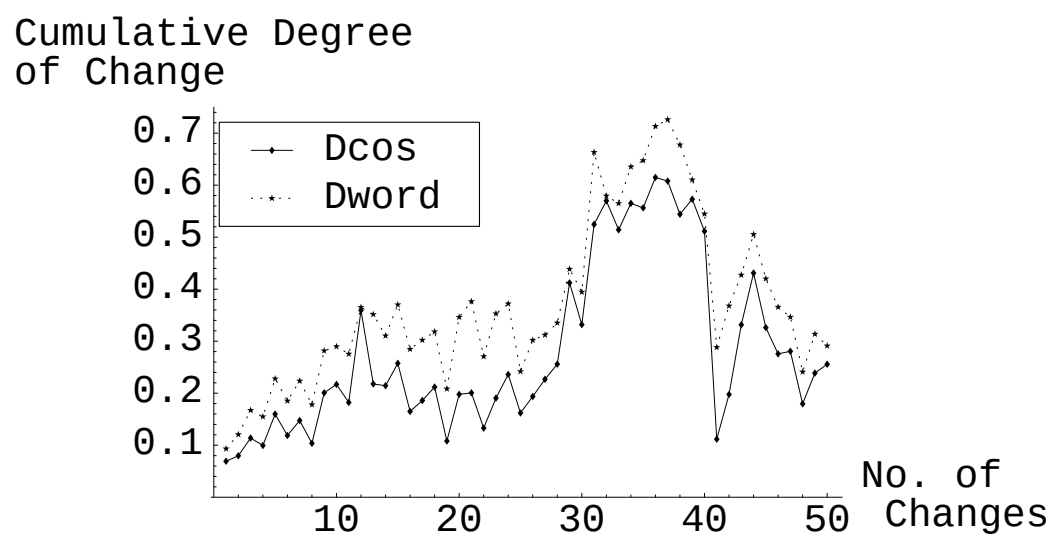


Figure 3.14: Relationship between *cumulative* degree of change and frequency of change.

updated on...” snippets (these are often generated automatically and in some cases do not coincide with any actual content modification). For the purposes of most search engines, changes such as these can safely be ignored.

In stark contrast, pages that underwent modifications between 30 and 40 times during our 51-week measurement period tended to exhibit a significant cumulative degree of change (above 50% on average), even though on average each modification only incurred a 5–10% degree of change. This discrepancy implies that although these pages tend to change only moderately with each modification, successive modifications often target different portion(s) of the page. As a result, the cumulative degree of change increases substantially over time. Pages in this category that are prone to experience the most substantive and durable alterations in content, even though the frequency of change is not the highest. From a search engine perspective, these moderately frequent changes are likely to be worthwhile to capture, whereas many of the very high frequency changes may not be, as suggested above.⁴ The question of how well the two classes of changes can be differentiated based solely on frequency of change statistics remains open.

3.5 Predictability of degree of change

As we concluded in Section 3.4, most of the changes detected in our experiments were very minor. Search engines may be able to exploit this fact by only re-downloading pages that have undergone significant revision since the last download. When resources are scarce it is important to avoid wasting resources on pages whose minor

⁴Note that [CG00c] suggests that search engines optimizing for overall freshness should, when resources are scarce, ignore high-frequency modifications so that resources can be used more profitably, even when all modifications are assumed to incur the same degree of change. Here, we are pointing out that many high-frequency modifications may be of little interest to search engines *intrinsically*, not just because resources can be saved by not incorporating them.

changes yield negligible benefit when incorporated into the search index. However, given the pull-oriented nature of the Web, the capability to differentiate between minor and major changes hinges on the ability to predict degree of change successfully. In this section we study the *predictability* of degree of change in Web pages. In particular, we seek to determine whether past degree of change is a good indicator of future degree of change, in terms of TF.IDF cosine distance. Our results obtained for the word distance metric were very similar so we omit them.

3.5.1 Overall predictability

We begin our analysis of predictability by studying the overall trends across all pages collected by our crawler. Later, in Section 3.5.2, we will extend our analysis to a finer granularity by inspecting individual sites. Figure 3.15(a) shows three scatter plots, each plotting cosine distance measured over a particular interval of time (one week, one month, and three months) on the horizontal axes. The vertical axes plot cosine distance measured over the ensuing time interval of the same duration. Each page contributes exactly one dot to each plot (although the vast majority are masked due to occlusion). These plots enable us to gauge the degree of correlation between successive time intervals. Points aligned along the diagonal (i.e. ones that satisfy the equation $y = x$) exhibit the same degree of change in successive measurement periods, and hence highly predictable degree of change. Therefore, our dataset manifests high predictability if most dots lie close to the diagonal.

To help us judge degree of predictability, we rank pages according to its straight-line distance from the diagonal ($y = x$) and divide them into four groups:

- **Group A:** The top 80% of pages in terms of proximity to the diagonal.
- **Group B:** Pages that fall between the top 80% and the top 90% in terms of

proximity to the diagonal.

- **Group C:** Pages that fall between the top 90% and the top 95% in terms of proximity to the diagonal.
- **Group D:** All remaining pages.

We plot each group using a different color in Figure 3.15: red⁵ for Group A, yellow for Group B, green for Group C, and blue for Group D. For example, from the red dots in left-most graph of Figure 3.15(c), we see that Group A (top 80% of pages) from `www.eonline.com` lie in a band between $y = x + 0.06$ and $y = x - 0.06$ from the diagonal. The narrower this band is, the more predictable the degree of change is. To make the “width” of each band easier to see, we plot the distance of members of each group from the diagonal (shown below each scatter plot; distance is normalized to the range $[0, 1]$).

It is clear from Figure 3.15(a) that most pages captured in our study change in a highly predictable manner, in terms of cosine distance. For example, from the yellow band (Group B, top 90%) in the second graph of Figure 3.15(a), we can see that for 90% of the pages, we can predict their future degree of change with $\pm 8\%$ error; yellow dots lie in the band $y = x \pm 0.08$. Not surprisingly, the degree of predictability decreases somewhat as we move to longer time intervals. The widths of the bands grow larger as the interval becomes longer. The fact that degree of change appears to be highly predictable, especially over the short term, is good news for search engines. By relying on simple predictive methods to estimate degree of change accurately, search engines may be able to take degree of change into account when selecting pages to redownload and reincorporate into the search index. As suggested by our results in

⁵In case this dissertation is viewed in grayscale, the reader should refer to the color legend at the bottom of Figure 3.15 for the color of each group of pages.

Section 3.4, doing so may enable search engines to use resources more effectively and ultimately achieve higher quality search results.

There are some caveats, however. First, as can be seen in Figure 3.15(a), the ability to predict degree of change accurately degrades mildly over time: the distance of every group from the diagonal grows over time. Second, a small but non-negligible fraction of pages defy even short-term prediction. Third, our conclusions are only valid over the pages considered in our study, which are drawn from popular Web sites. Further investigation will be necessary to determine whether our conclusions extend to less popular sites.

Fetterly et al. [FMN03] drew similar conclusions as to the predictability of degree of change, using number of changed shingles to measure change. They only studied short-term predictability over two successive week-long periods, however. Our results in Figure 3.15(a) show that even over a fairly lengthy span of time (successive three-month quarters), 80% of the pages are within a radius of 5% of the diagonal (recall that the diagonal represents exact predictability for cosine distance). It may not be feasible for certain search engines to monitor all pages of concern on a weekly basis, and the ability to rely on long-term predictability may help considerably with download scheduling and resource management.

3.5.2 Predictability for individual sites

Having shown that degree of change tends to be highly predictable for most pages included in our study, we turn to an investigation of predictability at the granularity of individual Web sites. After examining our data, we selected two sites that are representative of the range of site-level predictability present in our data: `www.columbia.edu` (an educational site) and `www.eonline.com` (an entertainment magazine site). Scatter plots of cosine distance predictability for the two sites are

shown in Figures 3.15(b) and (c). (For brevity we omit similar plots obtained for other sites.) Both sites exhibit good predictability overall, but the degree of predictability of pages from `www.columbia.edu` is significantly better than that of pages from `www.eonline.com`. Moreover, in the short term, pages from `www.eonline.com` tend to change much less predictably than the majority of pages in our overall study (Figure 3.15(a)). The degree of change of many pages on `www.eonline.com` either accelerated or decelerated between the first and second weeks, as is especially apparent for group D in the graphs. Perhaps this characteristic, which represents an outlier among the sites we studied, can be attributed to the fast-paced nature of trends and hot topics in the entertainment world.

From these examples we conclude that the ability to predict future degree of change from past behavior can vary a fair amount from site to site. (We confirmed that there is moderate variation in the general degree of predictability across other sites not discussed here due to space constraints.) Therefore, search engines may want to avoid heavy reliance on prediction for certain sites, and indeed for certain “rogue” pages that defy prediction even when other pages on the same site exhibit highly predictable behavior. Establishing reliable methods for identifying Web sites and individual pages for which prediction of degree of change is not likely to succeed (i.e., predicting predictability) is an important topic for future work.

3.6 Related Work

Others have studied Web evolution. We are not aware of any prior work on characterizing the evolution of the link structure of the Web experimentally. However, previous studies do touch upon aspects related to our measurements of the birth, modification, and death of individual pages over time. Here we discuss prior studies that exhibit some commonalities with our own.

In the first related study we are aware of to focus on degree of change, Lim et al. [LWP01] measured edit distance between two successive versions of around 6000 Web pages. More recently, Fetterly et al. [FMN03] repeatedly downloaded some 151 million Web pages and measured, among other things, degree of change by counting the number of changed “shingles.” The study of [FMN03] spanned a larger collection of pages than ours, but over a shorter period of time (eleven downloads over a period of roughly two months).

Aside from those differences, our study differs from [FMN03] in two significant ways. First, by recrawling sites from scratch each week, we were able to measure rates of web page creation (Fetterly et al. only measured deletion rates), which, interestingly, appear to match deletion rates closely. Second, our study concentrates specifically on aspects relevant to search engine technology, bringing out many implications for the design of search engine crawlers.

In particular, when measuring degree of change we focused on TF.IDF weighted cosine distance, which typically forms the basis for search engine ranking functions. Some of our results mirror those of [FMN03, LWP01] under a complementary distance metric, strengthening our shared conclusions. In addition, our work probes the following issues impacting search engine crawler design: We measured the correlation (or lack thereof) between frequency and degree of change. Furthermore, we studied the predictability of degree of change at a fine granularity, which turns out to vary significantly across domains. Finally, we measured the evolution of the hyperlink structure of the Web, which is a vital concern for modern search engines that combine link-based importance measures with traditional relevance scoring in their ranking functions [BP98]. Although there has been a rich body of theoretical work on Web growth models, e.g., [BKM00, CDK99, KRR00] to the best of our knowledge, our work is the first to study the evolution of Web link structure experimentally.

An earlier large-scale study of the evolutionary properties of the Web was performed by Brewington and Cybenko [BC00]. That study focused on page modification rates and times, and did not consider link structure evolution. A Boolean, “all or nothing” notion of page modification was used, in contrast to our study which measured degree of change in continuous domains. Using statistical modeling, [BC00] estimated the growth rate of the Web and determined the growth in the number of pages to be exponential, under the assumption of exponential growth in the number of Web hosts. A white paper from Cyveillance, Inc. [MM00] published in the same year also reported superlinear growth in the number of pages on the Web (although [MM00] does not reveal the methods used). These results are in opposition with our finding, which is based on analysis of our data in conjunction with new evidence of a stagnation in the growth rate of the number of Web hosts in recent years [ocl].

In [CG00a], lifespans and rates of change of a large number of Web pages were measured in order to assess the viability of adopting an “incremental” strategy for Web crawling. Changes were detected by comparing checksums, and was thus restricted to “all or nothing.” Degree of change was not measured. Earlier, Douglass et al. [DFK97] also studied Web page modification rates, gathering statistics useful from the point of view of designing an effective Web caching proxy. As a result of their focus on caching, the measurements of [DFK97] centered around the extent of replication of content across multiple pages and interactions between access patterns and modification patterns. Again, degree of change was not measured. Furthermore, neither [CG00a] nor [DFK97] measured page creation rates or link structure evolution.

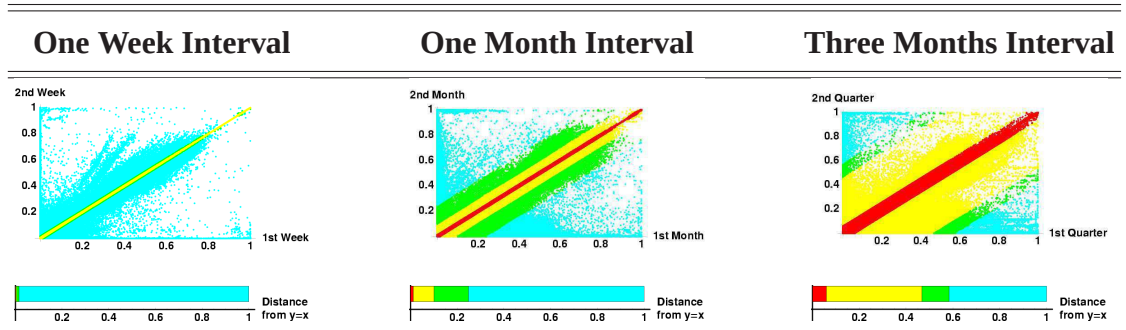
Pitkow and Pirolli [PP97] studied statistical characteristics of a single Web site for which they had access to user activity logs in addition to content snapshots. They characterized pages in terms of user access patterns and co-citation patterns. The co-citation analysis was performed over a static snapshot of the site—analysis of link

structure evolution was not undertaken. However, the evolution of individual pages in the site was studied, and correlations were found between frequency of modification and page lifetime, and between source of access (internal versus external) and page lifetime. Creation rates of new pages and degree of change were not measured in [PP97].

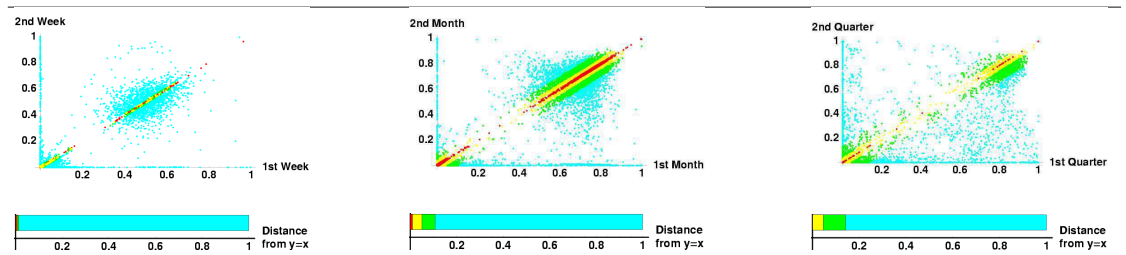
3.7 Summary

We have studied aspects of the evolving searchable Web over a one-year period that are of particular interest from the perspective of search engine design. Many of our findings may pertain to search engine crawlers, which aim to maximize search result quality by making effective use of available resources for incorporating changes. In particular, we found that existing pages are being removed from the Web and replaced by new ones at a very rapid rate. However, new pages tend to “borrow” their content heavily from existing pages. The minority of pages that do persist over extended periods of time typically exhibit very little substantive change (although many undergo superficial changes). For the exceptional pages that change significantly over their lifetimes, the degree of change tends to be highly predictable based on past degree of change. However, past *frequency* of change does not appear to be a good all-around predictor of degree of change.

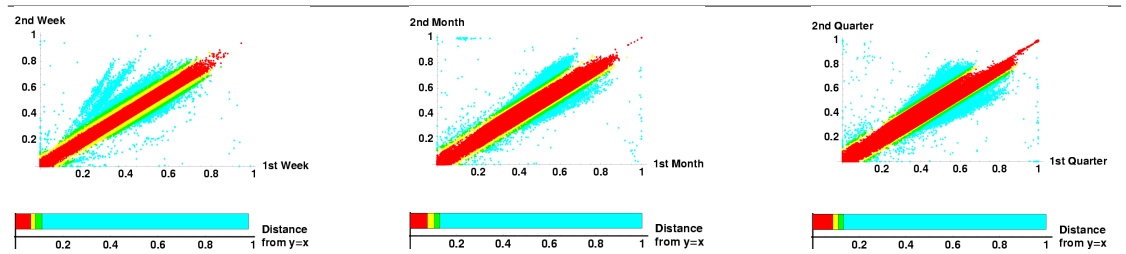
Since some search engines exploit link structure in their ranking algorithms, we also studied the evolution of links the Web. We determined that the link structure is evolving at an even faster rate than the pages themselves, with most links persisting for less than six months.



(a) All Web sites



(b) Web site `www.columbia.edu`



(c) Web site `www.eonline.com`

■ Group A (Top 80%)
 ■ Group B (Top 90%)
 ■ Group C (Top 95%)
 ■ Group D (Remainder)

Figure 3.15: (a) Cosine distance predictability over time for all pages, (b) only for the site `www.columbia.edu`, and (c) only for `www.eonline.com`. The first column represents changes falling one week apart, the second column shows changes one month apart, and the last column shows changes three months apart.

CHAPTER 4

Sampling-based Page Update

4.1 Introduction

A Web search engine typically downloads a large subset of the Web and indexes it to help users access Web pages. Since the pages on the Web are updated independently, it is necessary to periodically *poll* and *download* them in order to detect changed pages and incorporate the changes in our local index.

In order to avoid interference with the main tasks that the Web sites perform, search engines typically perform change detection and download in-batch (typically during off-peak hours) and at regular intervals. However, as the size of the Web grows, detecting changes and incorporating them to our local index becomes increasingly difficult. In particular, due to limited network and computational resources, it may not be feasible to check every Web page in the remote Web sites within a limited time window, and thus we may miss certain changes.

In this chapter, we address certain challenges that arise in this context: How we can detect and download as many changed Web pages as we can, when the Web pages are updated independently and when we are operating under limited resources? In this scenario, it is important exactly which Web page we decide to download and check, because we may waste a significant portion of our resources if we repeatedly download unchanged pages.

Our main idea to address this challenge is to use *sampling*. More specifically, we first download a small number of Web pages from each Web site which serve as *samples*. After that, we use the samples to decide which of the Web sites we should contact in order to download more Web pages.

In order to design and implement a good sampling-based policy, there are many important challenges to address. For example, how many samples should our crawler take from each Web site? Can a crawler improve effectiveness by dynamically adjusting the sample size? How should a crawler use the results from sampling? Can we combine a sampling-based policy with the state-of-the-art change-frequency-based [CG00d] policy? We address these challenges, in the following sections.

4.2 Framework

In this dissertation, we assume that the Web sites are updated *autonomously* and *independently* of our local copy. That is, we assume a *pull model* where the local copy needs to periodically check the Web sites to *detect* and *download* changes. This model is in contrast to a *push model* where the Web sites are *cooperative* and willing to *push* their updates to the local copies. Recently, Olston et al.[OW02] investigated the push model, but we believe the pull model may be more suitable for the applications within the realm of the World-Wide-Web.

We also assume that our local copy downloads Web pages *periodically in batch*, say, every weekend. That is, every weekend, we download a fixed number of Web pages from the Web sites and update our local copy. We call this interval – in this case one week – as a *download cycle*.

Our goal is to download as many changed Web pages as possible in each download cycle, using the same fixed download resources. This assumption is valid for the

World-Wide Web, since we maintain a large number of Web pages residing in many different sites, and we do not have enough resources to update them all in a short period of time. The following example illustrates a typical scenario that we assume.

Example 4.2.1 *We maintain local copies of 10 million Web pages downloaded from 10,000 sites. The 10,000 sites do not inform us of any changes, so we need to periodically download pages to detect and save changes in our copies. Since many users heavily access these pages during weekdays, we can download the pages only on weekends. Given our network bandwidth we can download up to one million pages every weekend. We want to use our limited download resources effectively so that we can download as many changed pages as possible in each week.*

4.2.1 Download policies

When we can download only a subset of Web pages in each download cycle, we need to carefully decide what Web page to download. There exist a multitude of ways for this decision, including the following:

1. *Round-robin:* We download Web pages in a round-robin fashion in each download cycle. In case of Example 4.2.1, for instance, we download the first 1 million pages in the first week, the second 1 million pages in the second week, etc. Because we maintain 10 million pages locally, every page will be updated exactly once every 10 weeks in this policy.
2. *Change-frequency-based:* Based on the past change history of a Web page, we estimate how often the page changes and decide how often to revisit the page. For instance, if we have downloaded a page once every month for one year, and if we detected 4 changes, we may estimate that the page changes once every 4 months and revisit the page accordingly. For more detailed description of this

policy, see references [CG00d, CLW98].

3. *Sampling-based*: We first sample a small number of Web pages from each Web site and estimate how many pages in that source have changed. We then allocate download resources to each Web site accordingly, based on the estimates. For instance, in case of Example 4.2.1, we may download 10 pages from each of the 10,000 Web sites as samples (a total of 100,000 page samples) and count how many pages in the samples have changed. (For now, we assume that we need to actually download a page to see whether the page has changed or not.) Then, based on the counts, we allocate the remaining 900,000 download resources to each Web site accordingly. Later in Section 4.3, we will discuss this policy in more detail.

The above three policies have their own merits and advantages. The round-robin policy is currently being used by many systems [BP98, HN99] due to its simplicity. It also guarantees that every Web page is downloaded at a regular interval. The frequency-based policy has the following advantages and disadvantages:

- **Advantage:** The frequency-based policy is proven to be optimal when we can estimate the change frequencies of Web pages accurately [CG00d].
- **Disadvantage:** 1) It is very difficult to estimate the change frequency of a Web page accurately. Unless we have a long change history of a Web page, existing estimation methods often lead to unreliable predictions [CG01], which in turn lead to an undesirable download policy. In addition, the change frequency itself may change over time, but we may not realize that it has changed.
2) In order to estimate the change frequencies, we need to keep track of the change history of every Web page. When we maintain a large number of pages, this tracking may incur significant storage and maintenance overhead

A sampling-based policy does not have the drawbacks mentioned above, because it makes a download decision purely based on the samples taken in the *current* download cycle. It does not need to keep track of the previous change history of Web pages. Later in Section 4.4, we will compare the effectiveness of the frequency-based policy and the sampling-based policy using real Web pages.

At this point, one may expect that a sampling-based policy would work only when the changes of the Web pages in the same source are *correlated*. However, we emphasize that this is not the case. If we can take *random* samples from each Web site, we are guaranteed that the fraction of changed pages in the samples is proportional – in a probabilistic sense – to the fraction of changed pages in the Web site. So a sampling-based policy does *not* assume any correlation between changes of Web pages. We should only be able to take *random* samples from each Web site.

We also note that it is possible to combine two or more policies to achieve desirable properties. For example, we may use half of our download resources in a round-robin fashion and use the remaining half for a sampling-based policy. This way, we can detect more changes than a simple round-robin policy, while downloading every item at least at a certain interval. Our study will help us employ a combined policy better, through better understanding of the sampling-based policy.

4.2.2 Evaluation metrics

In order to compare various download policies, we need an evaluation metric. We list three potential evaluation metrics in this subsection:

1. *ChangeRatio* metric: Informally, the *ChangeRatio* metric counts how many changed pages we download in a download cycle and uses this number as its performance. More precisely, the *ChangeRatio* metric is defined as the number

of downloaded *and* changed pages in a download cycle over the total number of downloaded pages in the cycle. For example, if we downloaded 1 million pages and detected 700,000 changed pages, the *ChangeRatio* is 0.7. Since the *ChangeRatio* may vary in different download cycles, we take its average over multiple download cycles. Our goal is to maximize the averaged *ChangeRatio*.

Note that in certain cases Web pages may have different “importance,” and we may want to detect more changes from more “important” pages. To formalize this notion, we may extend the simple definition of *ChangeRatio* by assigning weight w_i to each page o_i and define

$$ChangeRatio = \sum_{i \in \mathbf{R}} w_i \cdot \mathbf{1}(o_i) \quad (4.1)$$

Here, $\mathbf{1}(o_i)$ is an indicator function whose value is 1 when the page o_i has changed and 0 when it has not. $\mathbf{R}$ is the set of pages that have been downloaded. w_i ’s are normalized so that $\sum_{i \in \mathbf{R}} w_i = 1$. One can use the weights w_i to represent different types of importance of the pages such as their popularity or how critical their content is for a specific application (e.g. a news broker). When all w_i ’s are equal, this definition reduces to the simple definition.

The *ChangeRatio* metric is particularly useful when we want to store the *change history* of Web pages, such as for the WebArchive project [Web]. Because our goal is to store as complete change history as possible, we want to maximize the number of detected changes. A similar definition was used in [EMT01].

2. *Freshness* and *Age* metrics: In [CG00d], we proposed two other metrics, called *freshness* and *age*. The freshness of page o_i at time t is defined as

$$F(o_i; t) = \begin{cases} 1 & \text{if } o_i \text{ is up-to-date at time } t \\ 0 & \text{otherwise.} \end{cases} \quad (4.2)$$

(Up-to-date means that the locally stored image of the page is the same as the image at the source) and the freshness of the entire local copy at time t is

$$F(U; t) = \frac{1}{|U|} \sum_{o_i \in U} F(o_i; t). \quad (4.3)$$

Here, U is the set of all locally stored pages. Informally, the freshness metric represents the fraction of Web pages that are up-to-date. For example, if we maintain 100 pages and if 70 pages are up-to-date at t , its freshness is 0.7. Our goal is to maximize the *time-averaged* freshness under our resource constraints.

The second metric, the age of page o_i at time t , is defined as

$$A(o_i; t) = \begin{cases} 0 & \text{if } o_i \text{ is up-to-date at time } t \\ t - \text{modification time of } o_i & \\ \text{otherwise} & \end{cases} \quad (4.4)$$

and the age of the entire local copy is

$$A(U; t) = \frac{1}{|U|} \sum_{o_i \in U} A(o_i; t). \quad (4.5)$$

The age represents “how old” the local copy is. For example, if the source Web page changed one day ago, and if we have not downloaded the page since then, the age of our local page is one day. Our goal is to minimize the *time-averaged* age using limited resources. Similarly to the *ChangeRatio* metric, we

can incorporate different “importance” of objects, by assigning weight w_i ’s to pages and taking an weighted average.

The freshness and age metrics are suitable when we need to keep the local pages as up-to-date as possible. However, note that the metrics are hard to measure exactly in practice. That is, in order to estimate *exact* freshness (or age), we need to *instantaneously* compare *all* source pages to the local ones, which is often very difficult when we maintain a large number of Web pages. In addition, we want to optimize the *time-averaged* freshness and age values, but the time average can be obtained only when we know the *entire* change history of *every* Web page. Therefore, most of the studies on freshness and age are conducted through theoretical analysis, assuming some stochastic models for page changes.

3. *Divergence* metric: In [OW02], Olston et al. proposed a very general “staleness” metric called *divergence*. Intuitively, a divergence value represents how different a local Web page is from the source page. For example, in a stock-market-monitoring application – where we locally copy stock prices – we may define the divergence of a stock quote as the difference between its current price and the locally-stored value. In general, a divergence metric can be defined as any monotonically-increasing function [OW02].

In this dissertation, we mainly use the *ChangeRatio* as our evaluation metric. We made this choice because 1) it is easy to measure in practice on real data and 2) high *ChangeRatio* indirectly implies high freshness, low age, and less divergence. Also in the remainder of this chapter we mainly assume that pages have equivalent weights.

4.3 Sampling-based policies

In this section, we discuss sampling-based download policies in more detail. We start our discussion by clarifying our cost model for sampling.

4.3.1 Sampling cost model

A sampling-based policy needs to sample a few Web pages from each Web site in order to estimate how many pages in the site have changed. During sampling, we assume that we need to download an *entire Web page* to check whether the page has changed or not. That is, we assume that the cost for *sampling* a page is the same as the cost for actually *downloading* the page.

For example, if we can download 100,000 Web pages in each download cycle and if we sample a total of 10,000 Web pages, we can download 90,000 more Web pages in that cycle. We also assume that we do not need to download a sampled page again in the same download cycle, because the page was already downloaded during sampling.

This assumption makes our discussion simple, and it is straightforward to extend our current model to the case where sampling cost is lower than downloading cost. For instance, if sampling cost is only 10% of actual downloading cost, we may assume that we can download 99,000 ($= 100,000 - 0.1 \cdot 10,000$) more Web pages for the above scenario.

4.3.2 Greedy and proportional policies

We now discuss two sampling-based policies, *greedy* and *proportional*. To make our discussion concrete, we use the following as our running example.

Example 4.3.1 *We locally mirror two Web sites A and B. Each Web site has 100*

pages. We can download 100 pages every weekend. To estimate how many pages have changed, we sample 10 pages from each site. Out of the 10 samples, 7 pages changed in A and 2 pages changed in B . We need to decide how to allocate the remaining 80 ($= 100 - 2 \cdot 10$) page download resources to A and B . We assume that every page is equally important.

Given the sampling results, we may allocate the download resources to A and B either proportionally or greedily.

1. *Proportional policy*: We allocate the remaining resources to a site proportionally to its number of changed samples. That is, we download $80 \cdot \frac{7}{7+2} = 62$ pages from site A and $80 \cdot \frac{2}{7+2} = 18$ pages from site B .
2. *Greedy policy*: We start from the site that has the most changed samples and download all pages in the site. If we still have remaining download resources, we download more pages from the second-most changed site. We continue this process until we run out of download resources. In the above example, we use all 80 remaining resources for site A , because A has more changed samples than B .

In both policies, we allocate more download resources to the sites with more changed samples, hoping that we will detect more changes. While both policies are reasonable, we can see that the greedy policy is expected to yield better *ChangeRatio* than the proportional policy from the following simple analysis.

Probabilistically, $\frac{7}{10} = 70\%$ of the pages in site A would have changed and $\frac{2}{10} = 20\%$ of B pages would have changed. Therefore, the proportional policy is expected to detect $0.7 \cdot 62 + 0.2 \cdot 18 = 47$ changes from page downloads. Including the 9 ($= 7 + 2$) page changes detected during sampling, we detect 56 changes in total (i.e.,

the *ChangeRatio* is $56/100 = 0.56$). In contrast, the greedy policy is expected to detect $0.7 \cdot 80 + 9 = 65$ changes in total (*ChangeRatio* of 0.65).

In general, it is straightforward to prove that the expected *ChangeRatio* of the greedy policy is the highest among *all* sampling-based policies even in the case where the sites differ in sizes.

Theorem 1 (Optimality of Greedy Policy) *We sample the same number of random pages from each Web site and allocate remaining download resources based on the sampling results. In this scenario, the greedy policy is expected to give the highest ChangeRatio out of all sampling-based policies.* □

Proof For the proof of this and the remaining theorems of this chapter, please refer to Section 4.7. ■

The above theorem shows that the greedy policy yields better *ChangeRatio* on average than the proportional policy. However, the greedy policy may have *larger variation* in its performance, because the greedy policy aggressively allocates *all* of its resources to the site with more estimated changes. When the estimation is correct, this choice yields very high *ChangeRatio*, but when the estimation is incorrect, it also yields very low *ChangeRatio*. In contrast, the proportional policy downloads pages from *every* site, so even when the estimation is inaccurate, it still shows relatively high performance. Later in Section 4.4, we will investigate this issue experimentally.

4.3.3 Optimal sample size

In a sampling-based policy, the size of samples affects performance significantly. In this section, we study the optimal sample size that yields the highest *ChangeRatio*. In order to understand the impact of sample size, let us consider a scenario similar to Example 4.3.1, but now assume that we sample 50 pages from each site (instead of

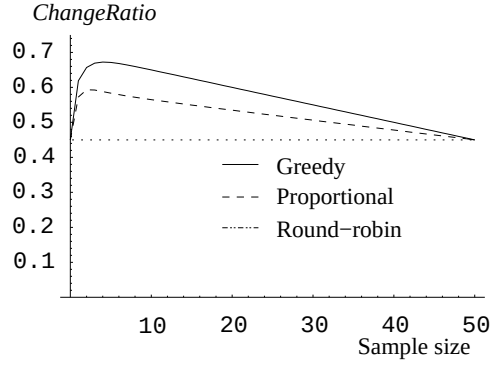


Figure 4.1: Expected *ChangeRatio* for various sample sizes

10). In this case, we use all of our 100 download resources just for sampling, so we cannot download any more pages from the site with more changed pages. Therefore, the performance of a sampling-based policy would be similar to that of the round-robin policy, because we download random pages during sampling. At the other extreme, if we sample only one or two pages from each site, there is a high chance that the estimated number of changes from samples is inaccurate and we make a wrong download decision.

Figure 4.1 illustrates this issue more precisely. We obtained the graph assuming that there are two sites, *A* and *B*, with 100 pages each, and we download 100 pages in one download cycle. We also assumed that 70 pages changed in site *A* and 20 pages changed in site *B*. The horizontal axis shows the sample size that a policy uses and the vertical axis shows the expected *ChangeRatio* of the round-robin, greedy and proportional policies at the given sample size. The graph was obtained analytically.

Note that the greedy and the proportional policies show the same expected *ChangeRatio*, 0.45, as the round-robin policy when the sample size is either 0 or 50. This is because when the sample size is 0, both policies select a random site for download, and when the sample size is 50, both policies use all their resources just for sampling. Also note that the proportional and the greedy policies show similar performance when the

sample size is small (≤ 2). This little difference is because the greedy policy is more likely to make an inaccurate download decision with small samples. It needs to sample “enough” pages to make a good decision. From the graph, we can see that the greedy policy shows the optimal performance when it samples about 5 pages from each site.

In general, we can derive the optimal sample size for the greedy policy analytically. To help derivation, we first introduce some notation.

We assume that all Web sites have the same number of Web pages, N . In practice, different Web sites may have different numbers of pages, but in this case, we may interpret N as the *average* number of pages in overall sites. We use r to represent the ratio of our download resources to the total number of pages that we maintain. For example, if we maintain 200 pages and if we can download 100 pages in each download cycle, r is 0.5. We use ρ_i to represent the fraction of changed pages in site S_i . For instance, if site S_1 has 100 pages and if 70 pages have changed, $\rho_1 = 0.7$. When sites have different ρ_i values, we can plot the histogram of ρ_i values as in Figure 4.2 and approximate it by a continuous density function $f(\rho)$. The goal of the greedy policy is to download pages only from the sites whose ρ_i values are the highest $100 \cdot r\%$ (say, the gray region in the figure). We use ρ_t to represent the threshold ρ value: The sites whose ρ values are higher than ρ_t belong to the top $100 \cdot r\%$ sites. We use $\bar{\rho}$ to represent the average ρ values over all sites. We use $\bar{\rho}_r$ to represent the average ρ values of the sites in the gray region (the sites whose ρ values are above ρ_t). In Table 4.1, we summarize our notation. Some of the notation in the table will be introduced later.

Under this notation, we can expect that the optimal sample size will depend on the distribution $f(\rho)$, our resource constraints, and the number of pages in the Web sites. The following theorem shows how these parameters affect the optimal sample size.

Theorem 2 (Optimal sample size) *The optimal sample size, s , under the greedy policy is approximately*

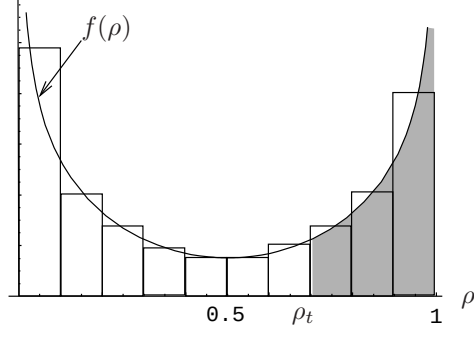


Figure 4.2: Histogram of ρ_i values of sites

Notation	Meaning
N	Average number of pages in all sites
R	Available download resources
r	Ratio of download resources to the total number of pages we maintain
s	Sample Size
k	Granularity (sample size) of adaptive policy
α	Confidence value of adaptive policy
ρ_i	Fraction of <i>changed</i> pages in site S_i
$f(\rho)$	Density function of all web sites' ρ values
ρ_t	Threshold ρ value. If $\rho_i > \rho_t$ for some S_i , then S_i belongs to the highest $100 \cdot r\%$ sites
$\bar{\rho}$	Average ρ values over all web sites
$\bar{\rho}_r$	Average ρ value of web sites having $\rho_i > \rho_t$

Table 4.1: Notation used throughout Chapter 4.

$$s \approx \sqrt{\frac{Nr f(\rho_t)}{6(\bar{\rho}_r - \bar{\rho})}} \quad (4.6)$$

□

Proof Please refer to Section 4.7. ■

Intuitively, we can understand the result of Theorem 2 as follows: First, when r is large (i.e., when we have relatively large download resources compared to the number of pages that we maintain) we can use more of our resources for sampling, because we can still download many pages from high ρ sites using the remaining resources. Second, when N is large (i.e., when Web sites have more pages), we need to sample more pages from the sites to predict their ρ values better.

Another factor, $\bar{\rho}_r - \bar{\rho}$, indicates that we can sample less pages when $\bar{\rho}_r - \bar{\rho}$ is high (i.e., when the ρ values of the top $100 \cdot r\%$ Web sites are much higher than the average ρ value.) This is because when the ρ values are very different among the sites, the estimated ρ_i values from samples will be very different, so it becomes easier to identify the high ρ sites from the others.

The final factor $f(\rho_t)$ indicates that we need to sample more pages when the value of the density function $f(\rho)$ is high at ρ_t . This is because when many Web sites have ρ values close to ρ_t (i.e., when $f(\rho_t)$ is large), it is more difficult to tell exactly which sites have ρ values higher/lower than ρ_t .

Using the formula in Theorem 2, we can estimate the optimal sample size when we know the distribution of ρ values. In certain cases, however, the distribution may be unknown, and we may not compute the optimal sample size accurately. Even in this scenario, we believe the result of Theorem 2 is still useful, because it shows that the optimal sample size s is proportional to the square root of Nr . As a rule of thumb, therefore, when we do not know the exact distribution of ρ values, we may use $\sqrt{Nr}$ as a rough approximation for the optimal sample size. Clearly, other factors are important

to determine the *exact* optimal size, but this approximation will be roughly in the same range as the optimal size, different only by a constant factor. Later in the experiment section, we will verify the result of this section using real Web data.

4.3.4 Adaptive sampling

The policies that we have discussed so far are *two-stage* policies. That is, we first take a fixed number of samples from each site at a *sampling stage*, and then we download more pages from high ρ sites at a *download stage*. Instead of a two-stage policy, we now discuss an adaptive sampling policy that tries to adjust the sample size dynamically and adaptively.

Our new adaptive policy is essentially based on the greedy policy: After sampling some pages from each site, if we are certain that the ρ value of a site is very high, we download all pages from the site. The difference is that the sample size is not determined in advance under the adaptive policy.

To illustrate, let us assume that we maintain local copies of 4 Web sites, S_1 through S_4 . Their ρ values are $\rho_1 = 0, \rho_2 = 0.45, \rho_3 = 0.55, \rho_4 = 1$, and each site has 100 pages. We can download a total of 200 pages in each download cycle. Roughly, our goal is to identify the two Web sites with high ρ values that we will download pages from.

Given the high ρ value of the site S_4 , we can expect that the samples from S_4 will have much more changes than the other samples. Therefore, it is relatively safe to pick S_4 for page download early in our sampling. Similarly, it is safe to discard S_1 early on, because of its low ρ value. Compared to S_1 and S_4 , S_2 and S_3 require larger samples, because their ρ values are similar and it is difficult to tell which one has a higher ρ value. Based on this intuition, we propose the policy described in Figure 4.3.

// ρ_i very high. We download pages from S_i

The algorithm takes two input parameters, α and k , whose intuition is given later. Roughly, the algorithm proceeds as follows: It samples k pages from each Web site, and based on the samples it estimates the ρ_i value and its $100 \cdot \alpha\%$ confidence interval for each site (Steps [3] through [6]). Given the distribution of the estimated ρ_i values, it can predict the threshold ρ_t value (Step [7]). For example, if we can download about half of the sites in each download cycle, and if half of the estimated ρ_i 's are above 0.6, $\rho_t = 0.6$.

After estimating ρ_t , it compares the confidence intervals of ρ_i to the threshold. If the confidence interval for S_i is strictly lower than the threshold ($h_i < \rho_t$), it stops sampling from the site (Step [9]); It has enough evidence that the ρ_i of S_i is below the threshold. Similarly, if the confidence interval of S_i is strictly above the threshold ($\rho_t < l_i$), it downloads all pages from the site (Step [10]).

The α and k values are configuration parameters set by the user. When the α value is low, the algorithm makes a download/discard decision “aggressively” and picks a site for download (or discard) even with low confidence. Thus, it allocates less resources to sampling and more resources to page downloads. The k value determines the granularity of sampling adjustment. When k is small, the algorithm re-estimates ρ_i values more frequently and makes a download (or discard) decision more often. Thus, the algorithm may show better performance but it may require more processing power. Later in Section 4.4, we will study the impact of α and k values on the effectiveness of the policy. We will try to identify good α and k values that yield high performance.

4.3.5 Subset sampling under low download resources

So far, we have implicitly assumed that we have a sufficiently large amount of download resources, so that we can sample a reasonable number of pages from each site and still download more pages from high ρ sites. In certain cases, however, this as-

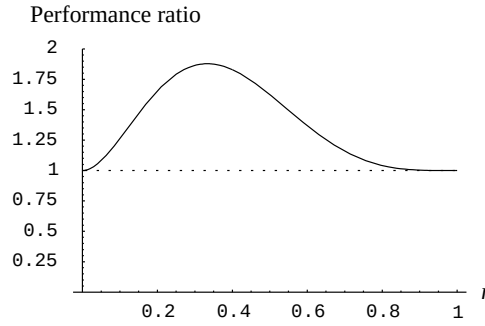


Figure 4.4: Comparison of the round-robin policy and a sampling policy for various resource constraints

sumption may not be valid. We may not be able to sample enough pages from each site, due to limited available resources. In this section, we study how we should handle low-resource scenarios.

Generally, there is an interesting relationship between the download resource size and the performance of a sampling-based policy. At one extreme, when we have few download resources and cannot sample enough pages from each site, a sampling-based policy would perform similarly to the round-robin policy: after sampling a couple of pages from each site, we cannot download any more pages from high ρ sites, and we end up visiting a small but different portion of the pages in each cycle, just like the round-robin policy. (Because we take random samples, we will visit different pages in different cycles with high probability.) At the other extreme, if we have enough resources to download every page in each download cycle, a sampling-based policy will perform similarly to the round-robin policy again, because both policies will download all pages in every cycle.

Figure 4.4 shows a *hypothetical* graph that illustrates this relationship. The horizontal axis shows the resource ratio r (the number of download resources to the number of total pages we maintain). When r is 1, we can download all pages in each download cycle, and when r is 0, we can download no page. The vertical axis shows the

performance ratio of a sampling-based policy to the round-robin policy (*ChangeRatio* of a sampling-based policy over *ChangeRatio* of the round-robin policy). When the sampling-based policy performs better than the round-robin policy, this ratio is higher than 1. In the graph, a sampling-based and the round-robin policies show similar performance (the performance ratio is 1) when resource ratio r is close to either 0 or 1, because of the reasons discussed above. In between these two extremes, a sampling-based policy shows better performance than the round-robin policy.

To improve performance for the scenario of very limited resources ($r \approx 0$), we propose that a sampling-based policy should select a small *subset* of its data in each download cycle, and sample and download pages only from the subset:

- *Subset sampling under low download resources:* When the download resources are too limited to sample enough pages from each site, we group the sites into m subsets. In each download cycle, we pick one subset and sample and download pages only from the sites in the subset. We revisit the subsets in a round-robin manner over multiple download cycles.

For instance, consider the following example:

Example 4.3.2 *We maintain local caches of Web pages from 1000 sites. Each Web site has 100 pages. Every weekend, we can download 500 pages in total. In this scenario, our simple greedy (or adaptive) policy cannot work effectively, because we can sample less than one page from each site.*

To handle this scenario, we may use the subset-sampling policy. First, we divide the sites into groups of, say, 10 sites. Every week, we select a group of 10 sites and sample, say 10 pages from each of the 10 sites. Assuming we use the greedy-policy, we can use the remaining 400 ($500 - 10 \cdot 10$) download resources to download pages from high ρ sites.

When we need to use the subset-sampling policy, one important question is how many sites we should put in each subset. Should we sample 10 sites in one download cycle, sampling all 1000 sites over 100 cycles? Or should we sample 20 sites in each cycle? The answer depends on the amount of available resources and the distribution of ρ values among the sites. Although we cannot derive a closed formula for the optimal number of sites to sample, we believe that the number should be determined such that we can download all pages from high ρ sites after sampling.

For example, if the ρ values follow the distribution of Figure 4.2, and if roughly 30% of the sites belong to the grey region (high ρ region), we should be able to download all pages from these top 30% sites in each download cycle. If our subset is too small and if we have to download pages from lower ρ sites (given our download resources), performance would degrade. If our subset is too large and if we “waste” most of our resources for sampling, performance would also suffer. Later in Section 4.4, we experimentally study the effectiveness of the subset-sampling policy.

4.3.6 Is *Greedy* too greedy?

While the greedy (and adaptive) policy can improve the overall *ChangeRatio*, it may be possible that some pages are never downloaded, because the policy downloads pages only from the high ρ sites. The following theorem proves that this is not the case.

Theorem 3 *When every page changes at some points of time, every page is eventually downloaded.* □

Proof Please refer to Section 4.7. ■

Although the theorem proves that every page will eventually be downloaded, it does not guarantee that pages are downloaded within a “reasonable” period of time. It also does not address the case when some of the pages does not change at all. In

our experiment section, we will study how often each page is downloaded under the greedy policy using real Web data. We also note that if it is important to download every page within a certain interval, we may decide to combine the round-robin policy with the greedy (or other sampling-based) policy. For example, we may want to use, say, 30% of download resources in a round-robin fashion and use the remaining 70% for the greedy policy.

4.4 Experiments

Following on from our theoretical analysis, we conducted a number of experiments in order to study the behavior and performance of the aforementioned policies. Most of our experiments were conducted on real data collected from the Web. The dataset contained 6-month change history of approximately 353,000 Web pages distributed among 252 Web sites and is described in more detail in [CG00b]. The data was collected by our crawler, which visited the Web sites once every month for a period of 6 months. Since changes could be detected only from the second visit (in the first visit, we do not know whether a page has changed or not), we had a total of 5 change history data for each page. Thus, our experiments could run up to 5 download cycles.

Compared to the real Web the size of our dataset and the number of download cycles are relatively small, but as we will see in the following sections it is enough to bring up the potential of the sampling policies. Also, when it is necessary to run experiments on a longer change history, we assumed that our 5 cycle data would repeat over time. That is, if we detected changes from a page in the 2nd and 5th cycles, we assumed that we detect changes in 7th, 10th, 12th, 15th cycles, etc. Our dataset is publicly available from our Web site [Web].

We should emphasize that our later experiments did *not* actually crawl and down-

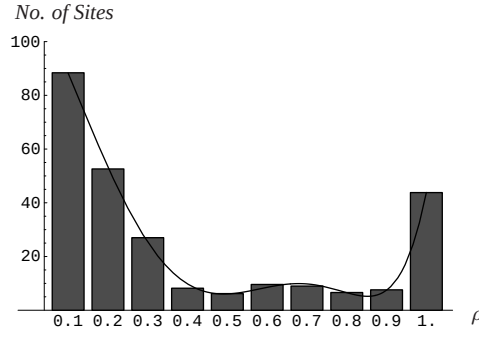


Figure 4.5: Histogram of ρ_i values in the dataset.

load pages. All experiments were conducted on the *same* data collected by our crawler. This setup enables a fair comparison among policies. Also, throughout experiments, we assume that the cost for sampling a page is the same as the cost for actually downloading it.

4.4.1 Distribution of ρ values

We start our discussion by investigating the properties of our dataset. In particular, we show the distribution of ρ values of the sites (Section 4.3.3) in Figure 4.5. The horizontal axis represents ranges of ρ values, and the vertical axis shows the number of Web sites with the given ρ value. Label 0.1 on the horizontal axis means the range of 0 to 0.1. Note that the ρ value of a site may vary between download cycles. However, we could not detect any meaningful fluctuation in ρ values between cycles from our dataset. We plotted the histogram using the *average* ρ value of a site over all 5 download cycles.

This figure shows that in our data, there exist quite a few Web sites whose pages change very frequently. About 18% of the sites has ρ values between 0.9 and 1. Also, a lot of sites are static and remain (almost) unaltered throughout our experiment. More than 35% of the sites has ρ values between 0 and 0.1 This fact intuitively suggests

that 1) it can be relatively easy to detect the high and low ρ sites using sampling and 2) if we can identify the Web sites with very high and low ρ values and allocate our download resources appropriately, we may observe a significant improvement in the number of detected changes.

While our data indicates that the ρ values of Web sites follow a V-shaped distribution, it will be also interesting how various download policies perform for different distributions. For this reason, we also ran some of our experiments on a *synthetic* data whose ρ values follow a *normal distribution*. The results on the synthetic data are reported in Section 4.4.8.

4.4.2 Rough comparison of download policies

In this section, we conduct a rough comparison of various policies using our real data. For the experiments, we assumed that each policy can download $R = 100,000$ pages in each download cycle. The greedy and proportional policies used sample size $s = 10$ and the adaptive policy used $k = 10$ (discussed in Section 4.3.4) and a confidence level $\alpha = 0.9$. Note that we did not try to optimize these parameters. We selected the numbers rather arbitrarily for this experiment. However, we believe that the results would show the relative potential of various algorithms. In later sections we will examine the impact of the various parameters more thoroughly.

Figure 4.6 shows the results. The horizontal axis corresponds to various policies and the vertical axis shows the *ChangeRatios* of the policies (averaged over 5 download cycles). From a first glimpse at this figure, the reader can observe that our greedy (Grd) and adaptive (Adp) policies perform surprisingly well compared to the round-robin (RR) and even the frequency-based (Frq) policy. Their *ChangeRatios* are almost twice as high as the frequency-based policy! Since their *ChangeRatios* are around 0.75, even if we could design a hypothetical *oracle* policy, which could magically download

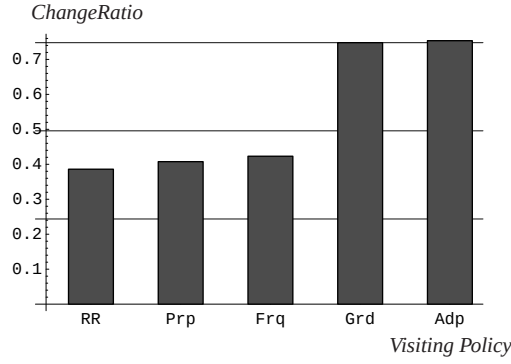


Figure 4.6: Comparison of the various download policies.

only changed pages, the improvement would be less than 25%. The performance of the proportional (Prp) policy is similar to that of the frequency policy, and the performance difference between the greedy and the adaptive policy is marginal.

While the results strongly indicate that the greedy and the adaptive policies are very effective, we note that the frequency-based policy could not show its full potential in this experiment, due to our small number of download cycles. Since the frequency-based policy did not know how often pages change, it visited *every* page *once* in a round-robin manner in the beginning, until the first half of the 4th cycle.¹ Only after that, the policy started to adjust revisit frequencies based on estimated change frequencies. Therefore, in the first three visits, the frequency-based policy showed the same performance as the round-robin policy and only from the second half of the 4th download cycle, it started to show some improvement.

Because of this fact, the comparison of the frequency-based policy and our sampling-based policies may not be fair, but we note that this is the situation in any practical system. Any system has to estimate page change frequencies in order to use the frequency-based policy, so it will suffer from poor performance in the beginning. In contrast, our sampling-based policies perform well without any change history data.

¹We had 353,000 pages and we visited 100,000 pages in each download cycle. Therefore, we need $3\frac{1}{2}$ cycles to visit every page once.

Later on, we will compare the long term performance of the frequency-based policy and our greedy policy.

4.4.3 Optimal sample size

In this subsection, we examine the impact of the sample size on the performance of sampling-based policies. For this purpose, we ran the greedy and the proportional policies on our data set, keeping the resource size constant to 100,000 pages and varying the sample size from 1 to 400. The outcome of this experiment is drawn in Figure 4.7. The horizontal axis represents the sample size and the vertical axis shows the *ChangeRatio* at the given sample size. From the graph, we can confirm the trend that we discussed before:

- When the sample size is too small, a sampling-based policy shows poor performance. It often makes a poor download decision. This degradation is particularly noticeable for the greedy policy.
- When the sample size becomes too large, performance also degrades, because sampling-based policies waste more resources for sampling than they ought to.

We can see that the optimal sample size for the greedy policy is around 10–50. This range matches well with the prediction of Theorem 2. In Section 4.3.3 we argued that $\sqrt{Nr}$ is a good rule of thumb for the optimal sample size when we do not know the exact distribution of ρ values. Given our parameters ($N = 353,000/252 \approx 1,400$ and $r = 100,000/353,000 \approx 0.28$), this formula predicts that the optimal sample size is $\sqrt{Nr} \approx 20$, which is in the range that we observe from our experiment.

From the graph we can see that for all sample sizes, the greedy policy shows better average *ChangeRatio* than the proportional policy. We expected this result from our discussion in Section 4.3.2, but we also discussed that the greedy policy may have a

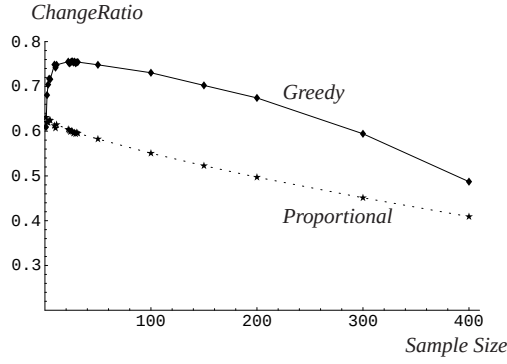


Figure 4.7: Performance of the greedy and proportional policies over various sample sizes.

larger variation in *ChangeRatio* than the proportional policy. To compare their variations, we measured the standard deviation (s.t.d.) of *ChangeRatio* between download cycles for both policies. From this estimation, we could see that the s.t.d. of the greedy policy is larger than that of the proportional policy (e.g., 0.027 vs. 0.023 for sample size 1). However, because the variation is very small (~ 0.02) compared to average *ChangeRatio* (~ 0.75), we believe that the variation issue is of negligible importance.

4.4.4 Resource size and subset sampling

We now study the effect of varying resource size on the performance of the greedy policy. For the experiments, we ran the greedy and the round-robin policies on our data set. The greedy policy used the sample size 10 and the resource size R varied from 3,000 to 300,000 pages.

Figure 4.8 shows the results from this experiment. The horizontal axis corresponds to the resource size and the vertical axis shows the *ChangeRatio* at the given resource size. The *oracle* policy is the one that can magically download only the changed pages. We show its performance for comparison purposes. Note that the *ChangeRatio* of the oracle policy goes below 1 for $R > 100,000$. This is because in each download cycle,

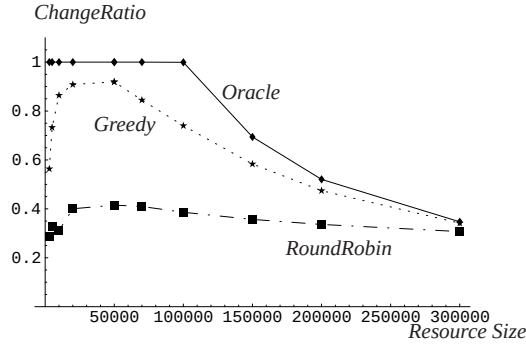


Figure 4.8: Performance of oracle, greedy and round-robin policies over various resource sizes.

only about 100,000 pages changed and if our resource size is larger than 100,000, the oracle policy starts to download unchanged pages. For most of resource sizes, the greedy policy shows much better performance than the round-robin policy.

The graph confirms our earlier discussion (Section 4.3.5): When the resource size is large, the performance of all policies become similar, because all policies download every page. When the resource size is too small, the performance of the greedy policy degrades. This degradation starts at $R < 20,000$.

To study the impact of the subset sampling policy, we divided sites into small subsets and sampled pages only from one subset in each download cycle when $R < 20,000$. The size of each subset was selected so that we can download about 18% of the pages in the subset in each download cycle. For example, when we have 10,000 download resources, each subset had about 60,000 pages. We selected 18% because about 18% of the sites in our dataset belonged to the right peak of the V-shaped distribution (Figure 4.5). The result from this experiment is shown in Figure 4.8. From the graph, we can see that the subset sampling policy improves the effectiveness of the greedy policy when the resource size is small. For instance, when $R = 5,000$ the *ChangeRatio* improves from 0.73 to 0.89 when we used the subset sampling policy.

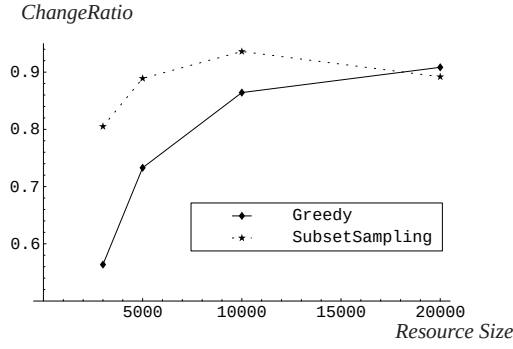


Figure 4.9: Comparison of *ChangeRatio* for *Greedy* and *SubsetSampling* for low number of resources.

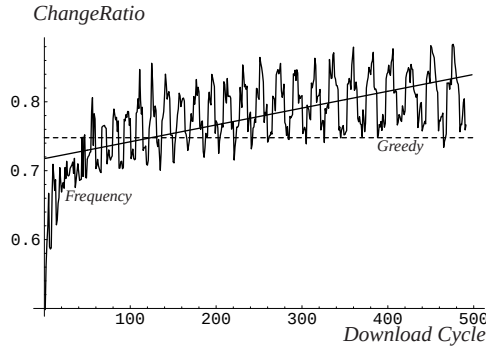


Figure 4.10: Performance of the frequency and the greedy policies over time

4.4.5 Long-term performance of the frequency policy

The results in Section 4.4.2 showed that the performance of the greedy policy is significantly better than the frequency-based policy in a short term. In this section, we study the long-term performance of the frequency-based policy and compare it to the greedy policy.

Towards this goal, we ran the frequency and the greedy policies for longer download cycles, by assuming that the observed change history of the pages repeats forever. For example, if we detected changes from a page in the 2nd and 5th cycles, we assumed that we detect changes in 7th, 10th, 12th, 15th cycles, etc. Figure 4.10 shows

the results. The horizontal axis corresponds to a download cycle, and the vertical axis shows the *ChangeRatio* at the given download cycle. The dashed line is the result of the greedy policy and the solid line with wide fluctuation is the result of the frequency-based policy.

The wide fluctuation in the frequency-based policy is mainly because it periodically downloads pages that rarely change. Even if a page has never changed, we cannot be sure that its change frequency is zero, so we have to periodically go and check the page for change. The dips in the graph correspond to the points when the policy downloaded infrequently changing pages. Note that the interval between these dips increases steadily over time. This is because as we accumulate more change history data, we can be more confident that the page does not change, and thus need to check the page less often.

From the graph, we can clearly see that the performance of the frequency-based policy improves steadily over time. Its performance is significantly lower than the greedy policy in the beginning, but from around 100th download cycle, it starts to show better performance. Therefore, in the long run, the frequency policy can be better than the greedy policy. However, keep in mind that 100 download cycles is a long period of time. Because we downloaded pages once every month, 100 cycles roughly correspond to 10 years!

4.4.6 Adaptive policy

We now study the impact of the k and α values (introduced in Figure 4.3) on the performance of the adaptive policy. To study their impact, we ran the adaptive-sampling policy for various k and α values. Figures 4.11 and 4.12 show the result. Figure 4.11 shows the *ChangeRatio* of the adaptive policy for various k values (the horizontal axis) when $\alpha = 0.9$. From the graph, we can see that the performance decreases as k in-

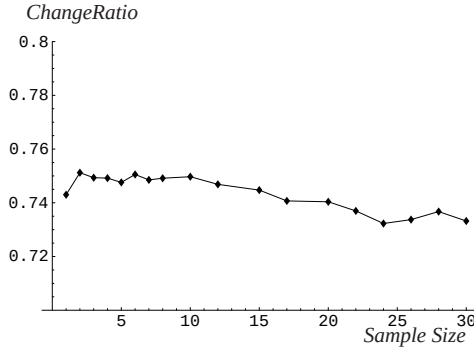


Figure 4.11: Performance of *Adaptive* over various sample sizes k .

creases. (There are small fluctuations, but we believe they are experimental variations.) This result is expected because when k is small, we try to re-estimate ρ_i values after small number of samples, and thus make a download decision more frequently with more accurate ρ_i values. From the figure, we can see that the performance decrease is relatively small until $k = 10$.

Figure 4.12 shows the *ChangeRatio* for various α values (the horizontal axis). From the graph, we can see that the confidence interval does not affect the performance of the policy significantly. We could not detect any meaningful difference in *ChangeRatio* for most of α values. One thing to note is that the performance for $k = 1$ is worse than others ($k = 5, 10$) when α is small ($\alpha \leq 0.75$). This is because when α is small and $k = 1$, the policy started to pick (or discard) a site for download after it took only 1 sample: Because it selected a site for download (or discard) too aggressively and too early, it often made wrong decisions. In other cases ($k = 5$ or 10 , or α is large), these early decisions did not happen, because the policy had to sample 5 or more pages when $k = 5$ or $k = 10$, or because it made a download (or discard) decision conservatively when α is large.

Based on the results, we believe $k = 10$ and $\alpha \approx 0.9$ are good parameters to use for a scenario similar to our Web data.

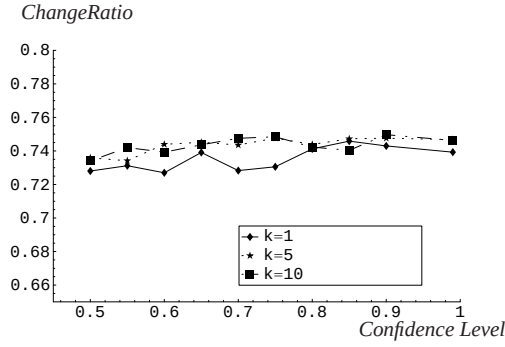


Figure 4.12: Performance of *Adaptive* over various confidence values α .

4.4.7 How greedy is *Greedy*?

In Section 4.3.6 we argued that the *Greedy* policy will eventually download all the pages within a Web site provided that those pages will change at some point of time. In this experiment we tried to evaluate how well the *Greedy* policy performs in this respect given our real data. More specifically we study 1) after i download cycles, what fraction of changed pages the *Greedy* policy manages to visit at least once? and 2) how often the *Greedy* policy downloads a page if the page changes, say, once every five cycles?

To answer the first question, we define a new metric $Fairness(i)$, as follows:

$$Fairness(i) = \frac{\text{no. of changed and visited pages up to } i\text{th cycle}}{\text{no. of total changed pages up to } i\text{th cycle}}$$

For example, if 50 pages have changed since the beginning, and we have revisited 30 of them so far, the *fairness* value is $30/50 = 0.6$. The intuition behind this metric is that a good download policy should revisit all changed pages at some point, and prevent *starvation* of revisits to some changed pages. Measuring *fairness* is particularly important for the *Greedy* policy, because it may allocate all its resources only to the sites with many changed pages due to its greedy nature.

Using our collection of real Web data we ran the *Greedy* policy using the same

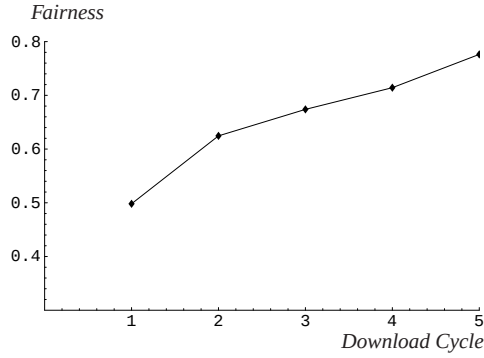


Figure 4.13: Fairness of *Greedy* policy.

parameters of Section 4.4.2, and plotted its *fairness* graph in Figure 4.13. In this graph the horizontal axis is the download cycle and the vertical axis represents the fairness value at the given download cycle. From this graph we can see that fairness is steadily increasing over time, reaching almost 0.8 after 5 download cycles. That is, only after 5 download cycles the *Greedy* policy have managed to visit 80% of the total changed pages in our dataset.²

Another interesting issue to study is how well the *Greedy* policy allocates its download resources to the pages of different change frequencies. Desirably, the *Greedy* policy should allocate more of its resources to the pages that are changing more often, in an attempt to detect more changes. To study this issue, we grouped the pages in our dataset according to the number of times they have changed during 5 download cycles, and for each group we measured how many times the pages were visited on average during the five download cycles. This result is shown in Figure 4.14. The horizontal axis shows the number of changes of a group and the vertical shows the average number of revisits for the group. From this plot we can clearly see that the *Greedy* policy allocates download resources in a proportional way among the groups, in the sense that it is visiting more often the pages that are changing more often.

²By *changed* here we mean the pages that have changed at least once during the 5 download cycles.

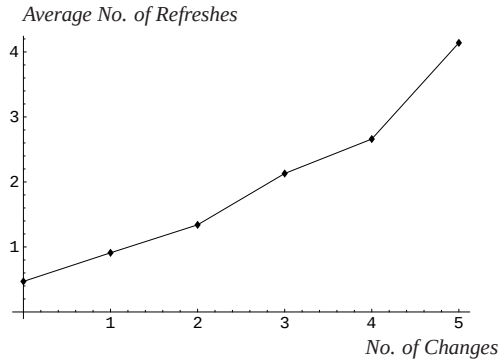


Figure 4.14: Average number of refreshes over number of changes for *Greedy* policy.

4.4.8 Performance under different distributions

Our previous experiments showed that the *Greedy* policy works well for the real Web data that follows a V-shaped ρ value distribution. It would be very interesting though, to examine how various policies would behave under different kinds of distributions. Towards this goal, we created synthetic data containing the change history of 300,000 pages. The pages are equally distributed among 300 web sites, so every site had 1,000 pages. The ρ values of these web sites were designed to follow a normal distribution with mean 0.5 and standard deviation 0.3.

Our main idea is to compare the performance of all the policies similarly to Section 4.4.2 using the synthetic data. As we will explain in detail shortly, the results from these experiments confirm that the general conclusions that we observed before are still valid for the normal distribution: While the exact numbers are different, the *Greedy* and *Adaptive* policies perform very well, and the *Frequency* policy behaves poorly in the beginning but manages to reach a similar performance to *Greedy* after a certain period of time.

In Figure 4.15, we show the performance of various policies under the synthetic data. For the experiment, we used the same parameters described in Section 4.4.2 and downloaded pages for 5 download cycles. Compared to Figure 4.6, all of algorithms

managed to have a high *ChangeRatio* value, because average ρ values were higher in the synthetic data. Again in this case, *Greedy* did very well with an overall performance reaching up to 0.82. The *Adaptive* policy also performed well, showing similar performance to that of the *Greedy*. The *Proportional* and *Frequency* policies showed similar performance to that of the *RoundRobin* policy. (Their performance is slightly worse than that of *RoundRobin*, but we believe it is due to statistical variations of our experiments.)

Frequency was not able to perform well in the experiment because, again, it ran only for five download cycles and did not have enough knowledge of the change history. To measure its long-term performance for this new distribution, we ran a longer experiment and observed that 1) *Frequency* improved its performance quickly over time and showed similar performance to *Greedy* after 10 download cycles, but 2) the final performance of *Frequency* was about the same as *Greedy* at around 0.82 (previously it was better than *Greedy*). We believe this result is because the average ρ value is high and all of the policies show relatively high *ChangeRatio* from the beginning. With high initial performance, *Frequency* can achieve only minor improvement, so it could reach its highest performance quickly, but the improvement was not very significant.

4.5 Related work

References [CG00d, CLW98] study how a crawler should download pages to maintain its index “up-to-date.” Assuming that the crawler knows the exact change frequencies of pages, the references present an optimal algorithm. As we learned from our experiments, this change-frequency-based algorithm performs relatively well once it collects a large amount of history data. However, history collection incurs significant overhead, and until it collects enough data, the algorithm performs poorly. Our sampling-based

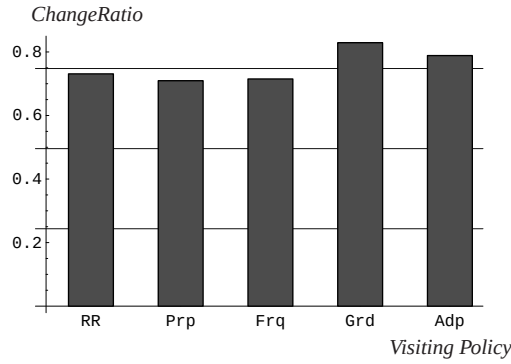


Figure 4.15: Comparison of various policies under a normal distribution $N(0.5, 0.3)$ of ρ values.

policies do not need to track any change history, and it shows significant improvement without any history data. Reference [EMT01] proposes another download algorithm based on linear programming. The algorithm shows promising results, but because the algorithm becomes more complex over time, the authors report that the algorithm has to periodically “reset” and “start from scratch;” The algorithm takes (practically) infinite amount of time to finish after a certain number of download cycles. In contrast, the complexity of our sampling-based algorithms stay the same over time.

A lot of work has been done to maintain the consistency of replicated data [BG84, ABG90, CKL97, KB94, LLS94]. This work studies the tradeoff between data consistency and read/write performance. In most of the existing work, however, researchers have assumed a *push* model, where the sources *notify* the replicated data of the updates. For example, Olston et al. [OW02] proposed a new architecture in which data sources can notify caches of important changes. In many contexts, particularly for the Web, this push model is not applicable, because data sources often do not inform others of their changes.

Sampling is a popular technique that has been used in multiple disciplines for various optimizations [HNS95, ZL94, WAA01, CMN99]. The contribution of this disser-

tation is to apply sampling techniques to the context of change detection, and study a variety of issues arising in this context.

The *multi-armed bandit problem* is well known in the statistics and AI community. The problem is to identify the slot machine with the highest chance of winning through *exploration* and *exploitation*. The problem is proven to be NP-hard [BF85], and people have proposed a range of approximation algorithms [ACF95]. The setting of the multi-armed bandit problem is slightly different from ours, because the bandit problem assumes that the user can play the best slot machine infinitely. In contrast, we can download only a limited number of pages from each Web site, so we need to find the top $r\%$ sources, not just the top source. This difference makes the policies take quite different forms.

4.6 Summary

We studied how we can detect changed Web pages effectively using sampling. We proposed three sampling-based policies, *greedy*, *proportional* and *adaptive*, and evaluated their performance analytically and experimentally. We also compared the sampling-based policies to other existing policies. Our experiments showed that the greedy policy is easy and simple to implement and shows one of the best performance in many scenarios. Given its simplicity and performance, we believe that the greedy policy is good for practical systems. Its complexity is similar to the widely-popular round-robin policy, while its performance is close to (or even better than) the frequency-based policy. Also, we learned that the frequency-based policy is not very effective in certain cases, because it takes a long time to estimate the change frequencies of pages.

4.7 Appendix: Proofs of the theorems

Theorem 1 We sample the same number of random pages from each data source and allocate remaining download resources based on the sampling results. In this scenario, the greedy policy is expected to give the highest *ChangeRatio* out of all sampling-based policies.

Proof We use c_i to represent the number of changed items in the samples from source S_i . Without losing generality, we assume c_1 is the highest and c_2 is the second highest and so on. We assume that we download R_i items from source S_i and the total of R items from all sources after sampling (i.e., $\sum_i R_i = R$). We also assume each data source has N_i data items.

When we take s random samples from every source and detect c_i changed items from those samples, we expect $100 \cdot \frac{c_i}{s} \%$ of the items in S_i to have changed. Therefore, we expect to detect $\frac{c_i}{s} R_i$ changes from S_i when we download R_i items from S_i . Then the total number of detected changes is expected to be

$$\sum_i \frac{c_i}{s} R_i, \quad \text{where } \sum_i R_i = R$$

Our goal is to maximize the above value. From the equation, it is easy to see that the formula takes its maximum value when we assign N_i to R_i for low i values ($i = 1, 2, \dots$), because c_i values are the highest for low i 's. That is, we have to start downloading data items from the sources with the most changed samples. ■

Theorem 2 The optimal sample size, s , under the greedy policy is approximately

$$s \approx \sqrt{\frac{Nr f(\rho_t)}{6(\bar{\rho}_r - \bar{\rho})}}$$

Proof To help our derivation, we first assume an *oracle greedy* policy, which can magically identify the sites with high ρ values and can directly download pages from

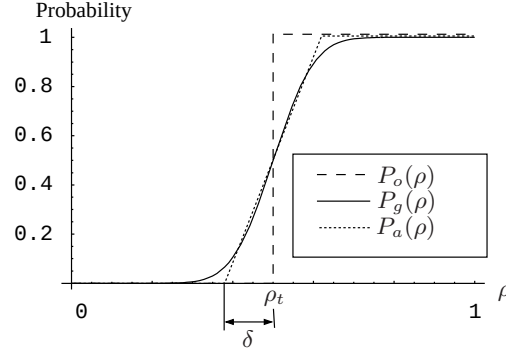


Figure 4.16: Probability $P_o(\rho)$ and $P_g(\rho)$

those sites without any sampling. We use $P_o(\rho_i)$ to denote the probability that the oracle greedy policy downloads a site S_i when its ρ value is ρ_i . By the definition of the oracle greedy policy, $P_o(\rho) = 1$ when $\rho > \rho_t$ and $P_o(\rho) = 0$ when $\rho < \rho_t$. We show the graph of $P_o(\rho)$ as a dashed line in Figure 4.16.

We use the $P_g(\rho_i)$ to represent the probability that our greedy policy downloads a site S_i when its ρ value is ρ_i . We show its graph as a solid line in Figure 4.16. Because the greedy policy sometimes makes wrong decisions, the graph $P_g(\rho)$ is not a step function like $P_o(\rho)$. It approaches 0 as ρ decreases, but when ρ is close to ρ_t , $P_g(\rho)$ is larger than 0 even if $\rho < \rho_t$ because some of the sites are mistakenly downloaded. Similarly, $P_g(\rho)$ approaches 1 as ρ increases, but when ρ is close to ρ_t , $P_g(\rho)$ is smaller than 1 even if $\rho > \rho_t$. As we increase the sample size s , the graph of $P_g(\rho)$ will become similar to that of $P_o(\rho)$, because the greedy policy makes more accurate download decision with larger samples.

We now approximate the function $P_g(\rho)$ by a piece-wise linear function $P_a(\rho)$

shown as a dotted line in Figure 4.16. The general form of $P_a(\rho)$ can be represented as

$$P_a(\rho) = \begin{cases} 0 & \text{for } \rho \in [0, \rho_t - \delta] \\ \frac{1}{2\delta}(\rho - \rho_t) + \frac{1}{2} & \text{for } \rho \in (\rho_t - \delta, \rho_t + \delta) \\ 1 & \text{for } \rho \in [\rho_t + \delta, 1] \end{cases} \quad (4.7)$$

and we need compute δ to find the approximation. Based on the result of Lemma 1 (which is given later), we will use $1/\sqrt{s}$ as the approximate δ value.

We now compute the expected performance of the greedy policy. Under the greedy policy, we download s sample pages from every site. Therefore, the expected number of changes that we detect during sampling is $M s \bar{\rho}$, where M is the total number of sites that we maintain and $\bar{\rho}$ is the average ρ value for overall sites. Once the greedy policy decides to download more pages from site S_i , the expected number of changes that we detect from S_i is $(N - s)\rho_i$. (Remember that we download $N - s$ more pages from S_i , because we do not download the sampled pages again.) Because the site S_i is downloaded with the probability $P_g(\rho_i)$ under the greedy policy, the total number of changes that we expect to detect during actual download is $\int_0^1 P_g(\rho)(N - s)\rho f(\rho)d\rho$. Therefore, the total number of changes that we detect both from sampling and from actual download is

$$\begin{aligned} & M(N - s) \int_0^1 P_g(\rho)\rho f(\rho)d\rho + M s \bar{\rho} \\ & \approx M(N - s) \int_0^1 P_a(\rho)\rho f(\rho)d\rho + M s \bar{\rho} \\ & = M(N - s) \left(\int_0^{\rho_t - \delta} P_a(\rho)\rho f(\rho)d\rho + \int_{\rho_t - \delta}^{\rho_t + \delta} P_a(\rho)\rho f(\rho)d\rho + \right. \\ & \quad \left. \int_{\rho_t + \delta}^1 P_a(\rho)\rho f(\rho)d\rho \right) + M s \bar{\rho} \end{aligned} \quad (4.8)$$

We now calculate the three integrals above:

$$\int_0^{\rho_t - \delta} P_a(\rho) \rho f(\rho) d\rho = 0, \text{ since } P_a(\rho) = 0 \text{ for } \rho \in [0, \rho_t - \delta] \quad (4.9)$$

Assuming that $f(\rho) \approx f(\rho_t)$ for $\rho \in (\rho_t - \delta, \rho_t + \delta)$:

$$\begin{aligned} \int_{\rho_t - \delta}^{\rho_t + \delta} P_a(\rho) \rho f(\rho) d\rho &= \int_{\rho_t - \delta}^{\rho_t + \delta} \left(\frac{1}{2\delta}(\rho - \rho_t) + \frac{1}{2} \right) \rho f(\rho_t) d\rho \\ &= \frac{1}{2\delta} f(\rho_t) \int_{\rho_t - \delta}^{\rho_t + \delta} \rho^2 - (\rho_t - \delta) \rho d\rho \\ &= \frac{1}{2\delta} f(\rho_t) \int_{\rho_t - \delta}^{\rho_t + \delta} \left(\frac{1}{3} \rho^3 + \frac{1}{2} (\delta - \rho_t) \rho^2 \right)' d\rho \\ &= \frac{1}{2\delta} f(\rho_t) \left(\frac{2}{3} \delta^3 + 2\delta^2 \rho_t \right) \\ &= f(\rho_t) \left(\frac{\delta^2}{3} + \delta \rho_t \right) \end{aligned} \quad (4.10)$$

For the third integral of Equation 4.8:

$$\begin{aligned} \int_{\rho_t + \delta}^1 P_a(\rho) \rho f(\rho) d\rho &= \int_{\rho_t + \delta}^1 \rho f(\rho) d\rho = \int_{\rho_t}^1 \rho f(\rho) d\rho - \int_{\rho_t}^{\rho_t + \delta} \rho f(\rho) d\rho \\ &= r \bar{\rho}_r - f(\rho_t) \delta \rho_t + \frac{1}{2} f(\rho_t) \delta^2 \end{aligned} \quad (4.11)$$

By replacing Equations 4.9, 4.10, 4.11 to Equation 4.8 and using Lemma 1 we can calculate that the total number of changes that we detect both from sampling and actual download is:

$$M(N - s) \left[\frac{f(\rho_t) \delta^2}{3} + f(\rho_t) \delta \rho_t + r \bar{\rho}_r - f(\rho_t) \delta \rho_t - \frac{1}{2} f(\rho_t) \delta^2 \right] + M s \bar{\rho} \quad (4.12)$$

Since we want to find the sample size s for which we detect the maximum number of changes we can consider Equation 4.12 as a function of s . It is straightforward to show that Equation 4.12 takes its maximum value for:

$$s = \sqrt{\frac{Nr f(\rho_t)}{6(\bar{\rho}_r - \bar{\rho})}} \quad (4.13)$$

■

Lemma 1 *The δ value for Equation 4.7 can be approximated as $1/\sqrt{s}$.* □

Proof From Figure 4.16, we see that δ is a value where $P_g(\rho_t + \delta)$ becomes close to 1. For this proof, we assume that we pick the δ value such that

$$P_g(\rho_t + \delta) = 0.95 \quad (4.14)$$

We now try to approximate $P_g(\rho_i)$, the probability that we download more pages from site S_i when its ρ value is ρ_i .

The site S_i is downloaded when its estimated ρ_i value (denoted as $\hat{\rho}_i$) is greater than ρ_t . That is, $P_g(\rho_i) = Pr\{\hat{\rho}_i > \rho_t\}$. To estimate the ρ_i value, we use X/s , where X is the number of changed pages in the sample and s is the sample size. That is, $\hat{\rho}_i = X/s$. Because we take s random samples from S_i where $100\rho_i\%$ of the pages changed, the random variable X follows a binomial distribution with a success rate ρ_i , and we can approximate $\hat{\rho}_i = X/s$ by the normal distribution with mean ρ_i and the standard deviation $\sqrt{\rho_i(1 - \rho_i)/s}$. When we do not know the ρ_i value, the standard deviation $\sqrt{\rho_i(1 - \rho_i)/s}$ is often approximated by $\frac{1}{2\sqrt{s}}$ [WMS97], so we may assume that $\hat{\rho}_i$ follows the normal distribution $N(\rho_i, \frac{1}{2\sqrt{s}})$.

When $\hat{\rho}_i$ follows this normal distribution, the probability $Pr\{\hat{\rho}_i > \rho_t\}$ is 0.95 when $\rho_t = \rho_i - 1.65\frac{1}{2\sqrt{s}}$. That is,

$$P_g(\rho_i) = 0.95 \text{ when } \rho_i = \rho_t + \frac{1.65}{2\sqrt{s}} \quad (4.15)$$

By comparing Equations 4.14 and 4.15, we can see that

$$\delta = \frac{1.65}{2\sqrt{s}} \approx \frac{1}{\sqrt{s}} \quad \blacksquare$$

Theorem 3 When every page changes at some points of time, every page is eventually downloaded.

Proof Let us select a random page that belongs to site S_i . Because every page changes, there exists a time t where all pages in S_i have changed. Then, in the download cycle after t , the site S_i will be downloaded under the greedy policy, because all sample pages in S_i have changed. $\blacksquare$

CHAPTER 5

Inverted Index Pruning with Correctness Guarantee

5.1 Introduction

The amount of information on the Web is growing at a prodigious rate [NCO04]. According to a recent study [GS05], it is estimated that the Web currently consists of more than 11 billion pages. Due to this immense amount of available information, the users are becoming more and more dependent on the Web search engines for locating relevant information on the Web. Typically, the Web search engines, similar to other information retrieval applications, are using a data structure called *inverted index*. An *inverted index* provides for the efficient retrieval of the documents (or Web pages) that contain a particular keyword.

In most cases, a query that the user issues may have thousands or even millions of matching documents. In order to avoid overwhelming the users with a huge amount of results, the search engines present the results in batches of 10 to 20 relevant documents. The user then looks through the first batch of results and, if she didn't find the answer she was looking for, she can potentially request to view the next batch. This process is repeated until the user either finds what she is looking for or gives up trying.

A recent study [JS03]¹ indicated that approximately 80% of the users examine at most the first 3 batches of the results. That is, 80% of the users typically view at most

¹See also articles at: <http://www.internetretailer.com/dailyNews.asp?id=11774>, and http://www.iprospect.com/media/press2004.04_20.htm (Accessed 02/15/2006).

30 to 60 results for every query that they issue to a search engine. At the same time, given the size of the Web, the inverted index that the search engines maintain can grow very large. Since the users are interested in a small number of results (and thus are viewing a small portion of the index for every query that they issue), using an index that is capable of returning all the results for a query may constitute a significant waste in terms of time, storage space and computational resources, which is bound to get worse as the Web grows larger over time [NCO04].

One natural solution to this problem is to create a small index on a *subset* of the documents that are likely to be returned as the top results (by using, for example, the pruning techniques in [CCF01, LS03]) and compute the first batch of answers using the pruned index. While this approach has been shown to give significant improvement in performance, it also leads to noticeable degradation in the quality of the search results, because the top answers are computed only from the pruned index [CCF01, LS03]. That is, even if a page should be placed as the top-matching page according to a search engine's ranking metric, the page may be placed behind the ones contained in the pruned index if the page did not become part of the pruned index for various reasons [CCF01, LS03]. Given the fierce competition among search engines today this degradation is clearly undesirable and needs to be addressed if possible.

In this chapter, we study how we can avoid *any* degradation of search quality due to the above performance optimization while still realizing most of its benefit. That is, we describe a number of simple (yet important) changes in the pruning techniques for creating the pruned index. We also describe a new answer computation algorithm that *guarantees* that the top-matching pages (according to the search-engine's ranking metric) are *always* placed at the top of search results, even though we are computing the first batch of answers from the pruned index most of the time. These enhanced pruning techniques and answer-computation algorithms are explored in the context of the *clus-*

ter architecture commonly employed by today’s search engines and we explain how search engines can minimize the operational cost of answering queries while providing high quality search results.

We will start our discussion by giving some background on the operation of an inverted index in Section 5.2. In Section 5.3, we study a cluster architecture of an inverted index that allows us to guarantee the correctness of results coming from a pruned index, and we discuss various issues of the performance of the architecture in Section 5.4. In Section 5.5, we examine various policies for reducing the size of the index; we perform an experimental evaluation of our policies in Section 5.7; Section 5.8 summarizes the related work and we summarize in Section 5.9.

5.2 Background

5.2.1 Overview of indexing and querying

Typically, a search engine downloads documents from the Web and maintains a local index that is used to answer queries. The task of the index is to answer very quickly which documents (or Web pages) contain some particular query term. This functionality is commonly implemented through the use of an inverted index:

Inverted indexes. Assume that we have already collected a set of documents² $\mathcal{D} = \{D_1, \dots, D_M\}$ and that we have extracted all the terms³ $\mathcal{T} = \{t_1, \dots, t_n\}$ from the documents. For every single term $t_i \in \mathcal{T}$ we maintain a list $I(t_i)$ of document IDs that contain t_i . Every entry in $I(t_i)$ is called a posting and can be extended to include additional information, such as how many times t_i appears in a document, the

²For the reader’s convenience, the notation used throughout the current chapter is summarized in Table 5.1.

³Here, for simplicity, we will consider as a term a sequence of consecutive non-whitespace characters in a document. Of course, there are cases where this definition might not be sufficient (such as Asian languages) but this is out of the scope of this dissertation.

Notation	Meaning
$tr(D, q)$	Textual similarity between document D and query q .
$tf(D, t_i)$	Term frequency of term t_i in document D .
$idf(t_i)$	Inverse document frequency of term t_i within the indexed collection of documents.
$pr(D)$	Query-independent quality of document D .
$r(D, q)$	Combined rank (textual similarity and query independent score) of document D for query q .
s	Size of pruned index.
$f(s)$	Fraction of queries that we can answer, given an index of size s .
$P(t_k)$	Fraction of queries that contain the term t_k .
$ I(t_k) $	Number of postings in inverted list I_k .
$ I_F $	Size of full index I_F .
$ I_P(s) $	Size of p-index for a given size s .
α	Weight of the textual part $tr(D, q)$ in the ranking function $r(D, q)$.

Table 5.1: Notation used throughout Chapter 5.

positions of t_i in the document, whether t_i is bold/italic, etc. The set of all the lists $\mathcal{I} = \{I(t_1), \dots, I(t_n)\}$ is our inverted index.

Answering a query. A query issued to our inverted index typically contains a subset of the index terms. We can represent a query as $q = \{t_1, t_2, \dots, t_w\}$. The goal of the search engine is to return the documents that are most relevant to query q . This is done in two steps: first we use the inverted index to find all the documents that contain the terms in the query. Second, once we have the relevant documents, we calculate the rank (or score) of each one of the documents with respect to the query and we return to the user the documents that rank the highest.

Most of the major search engines today return documents containing *all* query terms (i.e. they use AND-semantics). In order to make our discussions more concise, we will also assume the popular AND-semantics while answering a query. It is straightforward to extend our techniques to OR-semantics as well.

5.2.2 Ranking of the relevant documents

The exact ranking function that every search engine employs is a closely guarded secret. What is known however, is that every search engine bases the ranking of documents mainly on the following two different factors:

- **Query-dependent relevance.** This particular factor of relevance captures how relevant the query is to every document.

At a high level, given a document D , for every term t_i a search engine assigns a *term relevance score* $tr(D, t_i)$ to D . Given the $tr(D, t_i)$ scores for every t_i , then the *query-dependent relevance* of D to the query, noted as $tr(D, q)$, can be computed by combining the individual term relevance values.

For example, one popular way for calculating the query-dependent relevance is

to represent both the document D and the query q using the TF.IDF vector space model [SM83] and employ a *cosine distance* metric. In this case:

$$tr(D, t_i) = \frac{tf(D, t_i) \cdot idf(t_i)}{\sqrt{|D|}} \quad (5.1)$$

where $tf(D, t_i) = 1 + \log f_{D, t_i}$ (with f_{D, t_i} being the frequency of t_i in document D) and $idf(t_i) = \log(1 + M/f_{t_i})$ (with M being the total number of indexed documents, and f_{t_i} being the frequency of term t_i within the whole collection). Then we can compute the relevance of q to D as follows:

$$tr(D, q) = \sum_{i=0}^w \frac{tr(D, t_i)}{\sqrt{|q|}} \quad (5.2)$$

where w is the number of terms in the query. The higher the value of Equation 5.2, the higher the likelihood that D is relevant to q .

In practice, the exact form of $tr(D, t_i)$ may differ depending on the search engine. For example, a search engine may assign a higher $tr(D, t_i)$ value to D if t_i appears in the URL, anchor text or title of D . Similarly, the exact form of the query-dependent relevance $tr(D, q)$ may differ depending on the search engine. Thus, we will not restrict to any particular form; instead, we will make the generic assumption that the query-dependent relevance is computed as a function of the individual term relevance values in the query:

$$tr(D, q) = f_{tr}(tr(D, t_1), \dots, tr(D, t_w)) \quad (5.3)$$

- **Query-independent document quality.**

This is a factor that measures the overall “quality” of a document D independent of the particular query issued by the user.

For example, a popular technique that computes the general quality of a page is PageRank [PBM99] (or its variations such as HITS [Kle99]). Under PageRank, the quality of a page is essentially determined by its “link popularity” (i.e., how many other pages link to the page), and every page is assigned a global ranking score independent of the query that the user issued.

Another important factor that is often used, whose value is independent of a query, is the likelihood of being a “spam” page [NNM06, HMS02, Per01, GG05a, GG05b, GGP04]. Spam Web pages are the ones that are of little value to a human user and exist only to manipulate the search engine rankings. Most existing search engines compute the likelihood of being a spam page for every document D and give a low quality value (or a high “spam” score) to demote its ranking in the final result.

Here, we will use $pr(D)$ to denote the this query-independent part of the final ranking function for document D .

The final ranking score $r(D, q)$ of a document will depend on both the query-dependent and query-independent parts of the ranking function. The exact combination of these parts may be done in a variety of ways. In general, however, we can assume that the final ranking score of a document is a function of its query-dependent and query-independent relevance scores. More formally:

$$r(D, q) = f_r(tr(D, q), pr(D)) \quad (5.4)$$

For example, $f_r(tr(D, q), pr(D))$ may take the form $f_r(tr(D, q), pr(D)) = \alpha \cdot tr(D, q) + (1 - \alpha) \cdot pr(D)$, thus giving weight α to the query-dependent part and the

weight $1 - \alpha$ to the query-independent part.

In Equations 5.3 and 5.4 the exact form of f_r and f_{tr} can vary depending on the search engine. Therefore, to make our discussion applicable independent of the particular ranking function used by search engines, in this dissertation, we will make only the generic assumption that the ranking function $r(D, q)$ is monotonic on its parameters $tr(D, t_1), \dots, tr(D, t_w)$ and $pr(D)$.

Definition 1 A function $f(\alpha, \beta, \dots, \omega)$ is monotonic if $\forall \alpha_1 \geq \alpha_2, \forall \beta_1 \geq \beta_2, \dots \forall \omega_1 \geq \omega_2$ it holds that: $f(\alpha_1, \beta_1, \dots, \omega_1) \geq f(\alpha_2, \beta_2, \dots, \omega_2)$.

Roughly, the monotonicity of the ranking function implies that, between two documents D_1 and D_2 , if D_1 has higher query-dependent relevance than D_2 and also a higher query-independent score than D_2 , then D_1 should be ranked higher than D_2 , which we believe is a reasonable assumption in most practical settings.

5.3 Cluster architecture and cost savings from a pruned index

Search engines are accepting an enormous number of queries every day from eager users searching for relevant information. For example, Google is estimated to answer more than 250 million user queries per day.⁴ In order to cope with this huge query load, search engines typically replicate their index across a large cluster of machines as the following example illustrates:

Example 5.3.1 Consider a search engine that maintains a cluster of machines as in Figure 5.1(a). The size of its full inverted index I_F is larger than what can be stored in a single machine, so each copy of I_F is stored across four different machines. We also suppose that one copy of I_F can handle the query load of 1000 queries/sec. Assuming

⁴See <http://searchenginewatch.com/reports/article.php/2156461> (Accessed 16/02/2006).

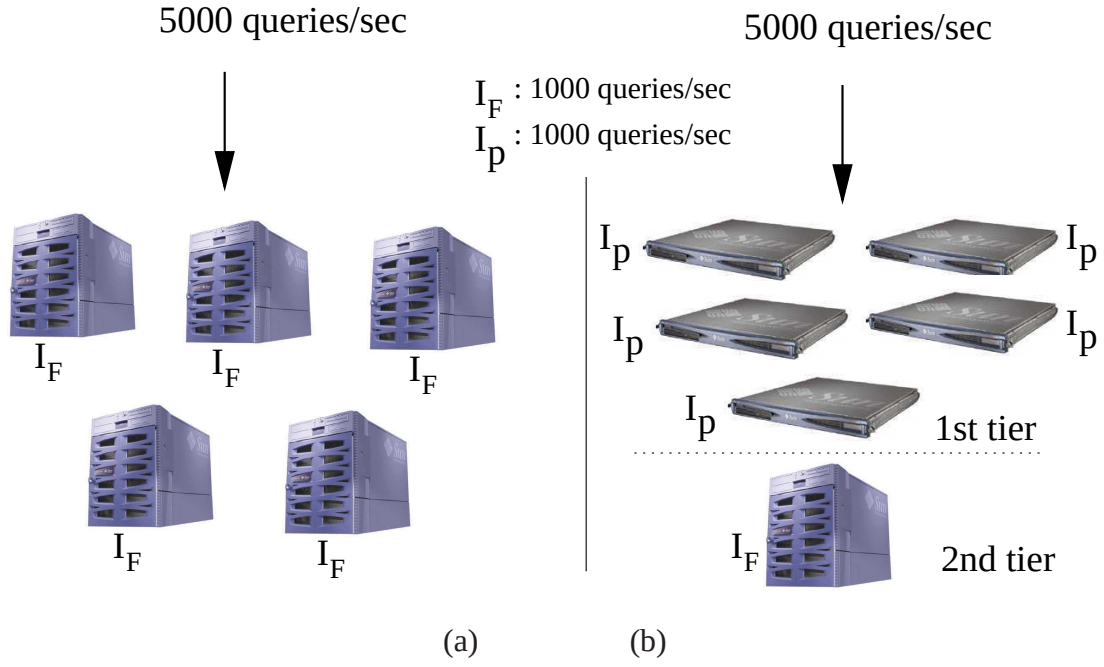


Figure 5.1: (a) Search engine replicates its full index I_F to increase query-answering capacity. (b) In the 1st tier, small p-indexes I_P handle most of the query. When I_P cannot answer a query, it is redirected to the 2nd tier, where the full index I_F is used to compute the answer.

that the search engine gets 5000 queries/sec, it needs to replicate I_F five times to handle the load. Overall, the search engine needs to maintain $4 \times 5 = 20$ machines in its cluster.

While fully replicating the entire index I_F multiple times is a straightforward way to scale to a large number of queries, typical query loads at search engines exhibit certain localities, allowing for significant reduction in cost by replicating only a small portion of the full index. In principle, this is typically done by pruning a full index I_F to create a smaller, *pruned index* (or *p-index*) I_P , which contains a subset of the documents that are likely to be returned as top results.

Given the p-index, search engines operate by employing a two-tiered index architecture as we show in Figure 5.1(b). All incoming queries are first directed to one of the p-indexes kept in the 1st tier. In the cases where a p-index cannot compute the answer (e.g. was unable to find enough documents to return to the user) the query is answered by redirecting it to the 2nd tier, where we maintain a full index I_F . The following example illustrates the potential reduction in the query-processing cost by employing this two-tier index architecture.

Example 5.3.2 *Assume the same parameter settings as in Example 5.3.1. That is, the search engine gets a query load of 5000 queries/sec and every copy of an index (both the full I_F and p-index I_P) can handle up to 1000 queries/sec. Also assume that the size of I_P is one fourth of I_F and thus can be stored on a single machine. Finally, suppose that the p-indexes can handle 80% of the user queries by themselves and only forward the remaining 20% queries to I_F . Under this setting, since all 5000/sec user queries are first directed to a p-index, five copies of I_P are needed in the 1st tier. For the second tier, since 20% (or 1000 queries/sec) are forwarded, we need to maintain one copy of I_F to handle the load. Overall we need a total of 9 machines (five machines for the five copies of I_P and four machines for one copy of I_F). Compared to Example 5.3.1, this is more than 50% reduction in the number of machines.*

The above example demonstrates the potential cost saving achieved by using a p-index. However, the two-tier architecture may have a significant drawback in terms of its result quality compared to the full replication of I_F ; given the fact that a p-index contains only a subset of the data of the full index, it is possible that, for some queries, the p-index may not contain the top-ranked document according to the ranking function used by the search engine and fail to return it as the top page, leading to noticeable quality degradation in search results. Given the fierce competition in the online search market, search engine operators desperately try to avoid any reduction in search quality

in order to maximize user satisfaction.

How can we avoid this potential degradation of search quality? Intuitively, we may avoid this problem by using the result computed from the p-index only if we *know* that it contains the top- k pages that are supposed to be returned in the first batch of results. Otherwise, we compute the answer from the full index I_F and return the correct top- k pages. We formalize this idea using the following the correctness indicator function and Algorithm 5.3.1.

Correctness guarantee. Given a query q , the p-index I_P returns the answer $\mathcal{A}$ together with a correctness indicator function $\mathcal{C}$. $\mathcal{C}$ is set to 1 if $\mathcal{A}$ is guaranteed to be identical to the result computed from the full index I_F . If it is possible that $\mathcal{A}$ is different, $\mathcal{C}$ is set to 0.

Algorithm 5.3.1 uses the correctness indicator function $\mathcal{C}$ to guarantee that the answer computed from the two-tier architecture is always identical to the answer from full I_F . As the input, the algorithm takes the keywords in the query together with the range of the answer to return to the user (e.g., top-10 answers, second top-10 answers, etc.). Given the input, the algorithm first tries to compute the answer from I_P and consults the value of $\mathcal{C}$ to see whether the computed answer is “correct.” If not, the algorithm recomputes the answer from the full index I_F and returns it to the user. Note that because the algorithm returns the answer from I_P only when $\mathcal{C} = 1$, its result is *guaranteed to be identical* to the result computed from the full index I_F . That is, there is no reduction in result quality while we benefit from the significant cost savings from the two-tier architecture.

There exist a number of important challenges in making Algorithm 5.3.1 work in practice. We now describe two of the most important questions that we address in this dissertation.

Question 5.3.1 *How much should we prune I_F to I_P to realize the maximum cost*

Algorithm 5.3.1 *Computation of answer with correctness guarantee*

Input $q = (\{t_1, \dots, t_n\}, [i, i + k])$ where

$\{t_1, \dots, t_n\}$: keywords in the query

$[i, i + k]$: range of the answer to return

Procedure

(1) $(\mathcal{A}, \mathcal{C}) = \text{ComputeAnswer}(q, I_P)$

(2) *If* $(\mathcal{C} = 1)$ *Then*

(3) *Return* $\mathcal{A}$

(4) *Else*

(5) $\mathcal{A} = \text{ComputeAnswer}(q, I_F)$

(6) *Return* $\mathcal{A}$

(7) *EndIf*

Figure 5.2: Computing the answer under the two-tier architecture with the result correctness guarantee.

savings?

In Example 5.3.2, we assumed that we prune I_F to one fourth of its size to get I_P . We also assumed I_P can handle 80% of the queries by itself at this size. Intuitively, there exists a tradeoff between the size of I_P and the fraction of queries that I_P can handle. When I_P is large and has more information, it will be able to handle more queries, but the cost for maintaining and looking up I_P will be higher. When I_P is small, the cost for I_P will be smaller, but more queries will be forwarded to I_F , requiring us to maintain more copies of I_F . Given this tradeoff, how should we determine the size of I_P in order to maximize the cost saving?

Question 5.3.2 *How should we prune I_F to I_P so that we can compute the correctness indicator function $\mathcal{C}$?*

The correctness guarantee of Algorithm 5.3.1 comes from our assumption that we can calculate $\mathcal{C}$ for the answer obtained from I_P . A straightforward way to calculate $\mathcal{C}$ is to compute the answer both from I_P and I_F and compare them. This naive solution, however, incurs a cost even higher than the full replication of I_F because the answers are computed *twice*: once from I_P and once from I_F . Then how can we know the correctness of the answer from I_P without computing the answer from I_F ?

We address the answers to the above two questions in the next few sections.

5.4 Optimal size of the p-index

In this section we study how we can determine how much to prune so that we minimize the cost of our two-tier index architecture. We start by considering the following simple example:

Example 5.4.1 Consider the full index I_F of Figure 5.1. If we prune I_F down to 20% we create p-index I_{P1} which can answer correctly 60% of the queries. If we prune I_F down to 40% we create p-index I_{P2} which can answer correctly 70% of the queries. Which one of the I_{P1} , I_{P2} is preferable?

Let us assume a query load of 5000 queries/sec and each machine handling 1000 queries/sec as shown in the architecture of Figure 5.1. We can calculate the size required for the overall architecture when using I_{P1} . We will need $\frac{5000}{1000} = 5$ machines for the p-indexes in the first tier. Since I_{P1} can answer correctly 60% of the queries, $(1 - 0.6) \cdot 5000 = 2000$ queries will be forwarded to the second tier. Therefore, we will need $\frac{2000}{1000} = 2$ full indexes for the second tier. Assuming that the size of a full index is $1TB$, and since the size of I_{P1} is 20% of I_F , overall we will need $5 \cdot 0.2 \cdot 1TB + 2 \cdot 1TB = 3TB$ of space for the two-tier architecture when we use I_{P1} . Similarly, we can calculate that we will need $3.5TB$ when we use I_{P2} . Therefore, I_{P1} is preferable because it uses less space.

The previous example demonstrates that the performance of the two-tier scheme depends on two parameters. First, it depends on the size of the p-index and second on the number of queries that it can directly handle without reverting back to the full index. We will use s to denote the size of the p-index relative to its full counterpart. For example, if $s = 0.2$, then the p-index is 20% of the size of I_F . If $s = 1$, then the p-index has the same size as the full index. We also use the notation $f(s)$ to denote the fraction of the queries that a p-index of size s can answer correctly. That is, when $f(s) = 0.3$, 30% of the queries can be computed from the p-index with the correctness indicator function $\mathcal{C} = 1$.

In the previous example we considered only two instances for the values of s and $f(s)$. In the general case, we can assume that there is a relation between these two values. For example, such a relation is depicted on the example graph of Figure 5.3.

Intuitively, when s is large, our p-index can handle more queries (i.e., large $f(s)$), and thus we look up the full index fewer times. However, in this case, because the p-index is large, the cost for a p-index lookup is also large. On the other hand, when s is small, the index-lookup cost for a p-index is small, but we need to revert back to the full index more often, increasing the overall cost of answering a query. Our goal is to handle the users' query load using minimum number of resources without any degradation in the search quality.

Given that Algorithm 5.3.1 guarantees no quality degradation, and given the fact that the number of machines needed to handle a query load is roughly proportional to the size of the overall indexes (which is proportional to the overall cost of running a search cluster), we can formalize our goal as follows:

Problem 5.4.1 (Optimal index size) *Given a query load Q and the function $f(s)$, the fraction of queries that the p-index can handle at a given size s , find the optimal index size s that minimizes the size of the overall indexes necessary to handle the load Q .*

We can show that we can determine the optimal index size based on the following theorem.

Theorem 4 *The cost for handling the query load Q is minimal when the size of the p-index, s , satisfies $\frac{df(s)}{ds} = 1$. □*

Proof For a given p-index size s , we can compute the size of the overall indexes necessary to handle the query load Q under the two-tier index architecture. We use $|Q|$ to represent the number of queries in the query load Q and use C to represent the number of queries that a single copy of an index can handle.

Consider $|I_F|$ the size of the full index and $|I_P(s)|$ the size of the p-index for a given size s . To handle Q , we need $\frac{|Q|}{C}$ of the first tier index. Thus, the space needed for the p-indexes is $\frac{|Q|}{C} \cdot |I_P(s)|$. Since the p-indexes can correctly handle only $f(s)$ fraction

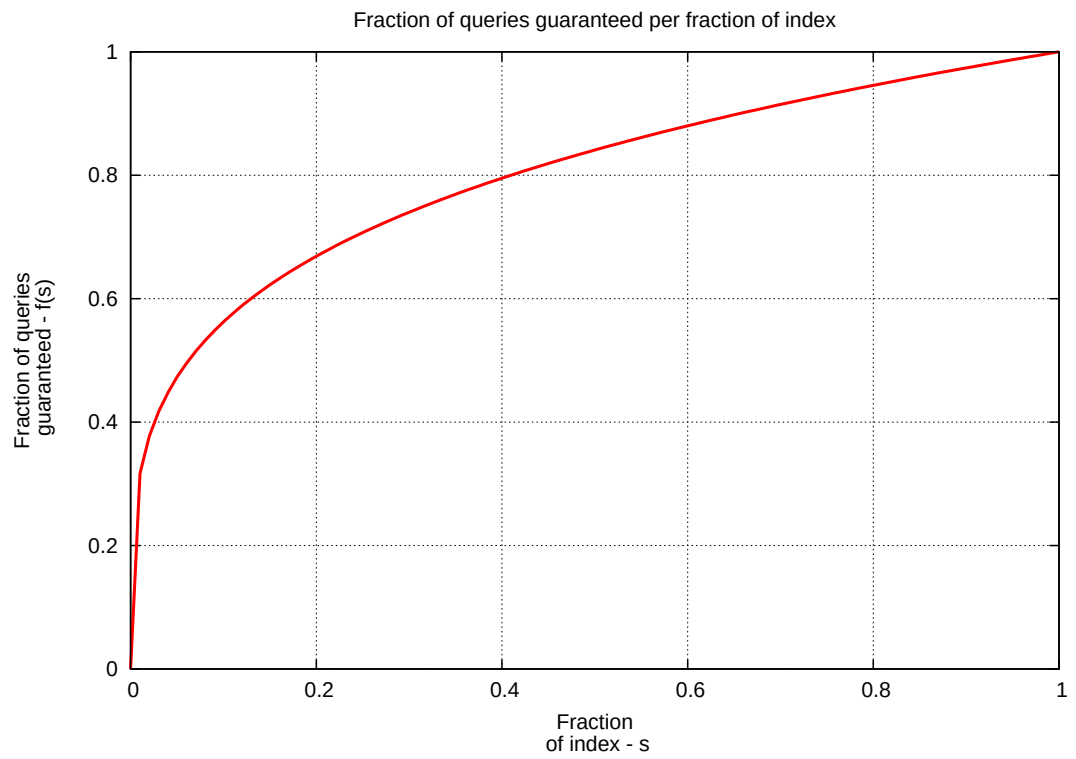


Figure 5.3: Example function showing the fraction of guaranteed queries $f(s)$ at a given size s of the p-index.

of queries, $|Q| \cdot (1 - f(s))$ queries are redirected to the second-tier full index. thus, $\frac{|Q| \cdot (1 - f(s))}{C}$ copies of the full index are needed, with the total size of $\frac{|Q| \cdot (1 - f(s))}{C} \cdot |I_F|$. Overall, we need the space of $\frac{|Q|}{C} \cdot |I_P(s)| + \frac{|Q| \cdot (1 - f(s))}{C} \cdot |I_F|$. Since for a given s , $|I_P(s)|$ is $s \cdot |I_F|$, the above equation can be simplified to $\frac{|Q|}{C} \cdot |I_F| (s + 1 - f(s))$, which is the total size of the two-tier indexes to handle the query load Q .

Since the cost is proportional to the size of the p-index, we can simply minimize the size of the p-index. That is, we want to minimize $\frac{|Q| \cdot |I_F|}{C} \cdot (s + 1 - f(s))$. For this, $\frac{d}{ds} \frac{|Q| \cdot |I_F|}{C} \cdot (s + 1 - f(s)) = 0 \Rightarrow \frac{df(s)}{ds} = 1$. ■

In our experiments, we computed the optimal index size s for the pruning policies that we explore in this chapter and found that when the p-index size is roughly around 20% we get the most significant reduction in the overall index size.

We should note that in the above problem formulation, we assume that the function $f(s)$ is concave and thus we can identify the *minimum* size from Theorem 4. As we will demonstrate later in Section 5.7, this concaveness assumption indeed holds in a practical scenario. Additionally, we assume that we know the user's query load Q in advance and thus we can obtain $f(s)$. Of course, predicting the exact query load issued by the users may not be straightforward. In Section 5.6 we discuss this issue and we show that the query load issued in the recent past is a good approximation of the expected query load in the near future.

We now focus on answering Question 5.3.2, i.e. how should we prune I_F so that we can compute $\mathcal{C}$.

5.5 Pruning policies

In this section, we now discuss how we should prune the full index I_F to I_P , so that we are able to compute the correctness function $\mathcal{C}$ from I_P itself (without computing

the full answer from I_F) while handling a large fraction of user queries from I_P . In designing the pruning policies, we observe the following two localities in the users' search behavior:

1. **Keyword locality:** Although there are many different words in the document collection that the search engine indexes, a few popular keywords constitute the majority of the query loads. This *keyword locality* implies that the search engine will be able to answer a significant fraction of user queries accurately even if it can handle only these few popular keywords.
2. **Document locality:** Even if a query has millions of matching documents, users typically look at only the first few results [JS03]. Thus, as long as search engines can compute the first few top- k answers correctly, users often will not notice that the search engine actually has not computed the correct answer for the remaining results (unless the users explicitly request them).

Based on the above two observations, we now investigate two different types of pruning policies: (1) a *horizontal pruning* policy, which takes advantage of the keyword locality by pruning a number of whole inverted lists from I_F , and (2) a *vertical pruning* policy, which takes advantage of the document locality by pruning individual postings from a set of inverted lists.

5.5.1 Horizontal pruning

One way to take advantage of the keyword locality is to create a p-index by pruning the inverted lists that correspond to the least frequent terms. We will refer to this method of pruning as “horizontal pruning,” since we are pruning the inverted index I_F “horizontally” as we show in Figure 5.4. The figure shows the list of documents containing

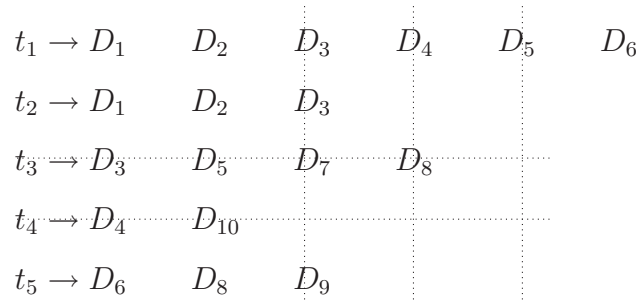


Figure 5.4: Horizontal and vertical pruning. During horizontal pruning, we remove inverted lists from the index: inverted lists t_3 and t_5 are removed. During vertical pruning we remove postings from the inverted lists: postings D_4, D_5, D_6 from t_1 and posting D_8 from t_2 are removed.

each term t_i , where we horizontally prune inverted lists for t_3 and t_5 , assuming that the terms do not appear frequently in the query load.

In general, the goal of horizontal pruning can be expressed as follows:

Problem 5.5.1 (Optimal horizontal pruning) Given the query load Q^5 and a goal index size $s \cdot |I_F|$ for the pruned index, select the inverted lists $I_P = \{I(t_1), \dots, I(t_h)\}$ such that $|I_P| \leq s \cdot |I_F|$ and the fraction of queries that I_P can answer (expressed by $f(s)$) is maximized.

Unfortunately, we can prove that the optimal solution to the above problem is intractable.

Theorem 5 The problem of calculating the optimal horizontal pruning is NP-hard. $\square$

Proof Assume a simplified version of our problem where $S = \{I(t_1), \dots, I(t_n)\}$ is the set of all the inverted lists inside the full index I_F and we are given a set of queries $Q = \{q_1, \dots, q_m\}$ with all queries being single-keyword queries (i.e., $q_i = \{t_i\}$). Assume that f_i is the fraction of queries in Q with the term t_i inside. Then our goal is to

⁵Again, our assumption is that we know the query load Q beforehand, which may not be possible in a practical scenario. In Sections 5.6 and 5.7 we show that the query load remains relatively stable and that we can achieve good performance by using past query loads to perform horizontal pruning.

Algorithm 5.5.1 Greedy horizontal pruning HS **Procedure**

- (1) $\forall t_i$, calculate $HS(t_i) = \frac{P(t_i)}{|I(t_i)|}$.
- (2) Include the inverted lists with the highest $HS(t_i)$ values such that $|I_P| \leq s \cdot |I_F|$.

Figure 5.5: Approximation algorithm for the optimal horizontal pruning.

select the subset of the inverted lists $S^* \subset S$ such that $\sum_{I(t_i) \in S^*} f_i$ is maximized under the constraint that $\sum_{I(t_i) \in S^*} |I(t_i)| \leq s \cdot \sum_{I(t_i) \in S} |I(t_i)|$. This is equivalent to the knapsack problem where the items are $I(t_i)$'s and their (profit, cost) pairs are $(f_i, |I(t_i)|)$. ■

Given the intractability of the optimal solution, we need to resort to an approximate solution. A common approach for similar knapsack problems is to adopt a greedy policy by keeping the items with the *maximum benefit per unit cost* [CLR01]. In our context, the potential benefit of an inverted list $I(t_i)$ is the the number of queries that can be answered by I_P when it includes $I(t_i)$. We approximate this number by the fraction of queries in the query load Q that include the term t_i and represent it as $P(t_i)$. For example, if 100 out of 1000 queries contain the term `computer`, then $P(\text{computer}) = 0.1$. The cost of including $I(t_i)$ in the p-index is its size $|I(t_i)|$. Thus, in our greedy approach in Figure 5.5, we include $I(t_i)$'s in the decreasing order of $P(t_i)/|I(t_i)|$ as long as $|I_P| \leq s \cdot |I_F|$.

Note that after horizontal pruning, if all keywords $\{t_1, \dots, t_n\}$ in the query q appear in I_P , the p-index has the same information as I_F as long as q is concerned. Therefore, as long as all keywords in q appear in I_P , the answer computed from I_P is guaranteed to be the same as the answer computed from I_F . Figure 5.6 formalizes this observation

Algorithm 5.5.2 *Computation of $\mathcal{C}$ for horizontal pruning***Procedure**

- (1) $\mathcal{C} = 1$
- (2) *Foreach* $t_i \in q$
- (3) *If* ($I(t_i) \notin I_P$) *Then* $\mathcal{C} = 0$
- (4) *Return* $\mathcal{C}$

Figure 5.6: Result guarantee in horizontal pruning.

and computes the correctness indicator function $\mathcal{C}$ for a horizontally pruned index I_P .⁶ It is straightforward to prove that the answer from I_P is identical to that from I_F if $\mathcal{C} = 1$ in the above algorithm.

5.5.2 Vertical pruning

At a high level, *vertical pruning* tries to take advantage of the observation that most users are mainly interested in viewing the top few answers to a query. Given this, it may be unnecessary to keep all postings in an inverted list $I(t_i)$, because users will not look at most of the documents in the list anyway. We depict the conceptual diagram of the vertical pruning policy in Figure 5.4. In the figure, we “vertically prune” postings corresponding to D_4, D_5 and D_6 of t_1 and D_8 of t_3 , assuming that these documents are unlikely to be part of top- k answers to user queries. More formally, the goal of a vertical-pruning policy can be expressed as follows:

Problem 5.5.2 (Optimal vertical pruning) *Given the query load Q and a goal index*

⁶When the inverted list for t_i does not exist in I_P , it will be helpful if we can distinguish the cases (1) when the inverted list exists in I_F but has been pruned in I_P and (2) when no documents in the collection contains the term t_i . This can be easily done by keeping an empty list for t_i in I_P if it has been pruned. For simplicity, we do not consider this optimization in the later experiment section.

Algorithm 5.5.3 *Global vertical pruning* VS_G

Procedure

- (1) Sort all documents D_i based on $pr(D_i)$
- (2) Find the threshold value τ_p , such that
only s fraction of the documents have $pr(D_i) > \tau_p$
- (4) Keep D_i in the inverted lists if $pr(D_i) > \tau_p$

Figure 5.7: Global vertical pruning based on pr .

size $s \cdot |I_F|$ for the pruning index, select a set of postings to remove from each $I(t_i)$, such that $|I_P| \leq s \cdot |I_F|$ and $f(s)$ is maximized.

We now discuss a few approaches to the vertical pruning problem. We first investigate the approach that is common with existing search engines and explain its biggest shortcoming: *it is not possible to compute the correctness function under this approach*. We then propose simple yet essential changes to the pruning policy that allow us to compute the correctness function.

5.5.2.1 PR-based pruning

This is the pruning policy that is commonly used by existing search engines. The basic idea is that the query-independent quality score $pr(D)$ is a very important factor in computing the final ranking of the document (e.g. PageRank is known to be one of the most important factors determining the overall ranking in the search results).

Therefore, we build the p-index only for the pages whose pr values are high (i.e., when $pr(D) > \tau_p$ for a threshold value τ_p). The hope is that most of the top-ranked results are likely to have high $pr(D)$ values, so the answer computed from this p-index

Algorithm 5.5.4 Local vertical pruning VS_L

N : maximum size of a single posting list

Procedure

- (1) *Foreach $I(t_i) \in I_F$*
- (2) *Sort D_i 's in $I(t_i)$ based on $pr(D_i)$*
- (3) *If $|I(t_i)| \leq N$ Then keep all D_i 's*
- (4) *Else keep the top- N D_i 's with the highest $pr(D_i)$*

Figure 5.8: Local vertical pruning based on pr .

is likely to be similar to the answer computed from the full index. Figure 5.7 describes this pruning policy more formally, where we sort all documents D_i 's by their respective $pr(D_i)$ values and keep a D_i in the p-index when its $pr(D_i)$ value is higher than the *global threshold* value τ_p .

Variations of this pruning policy are common among search engines today. One such variation is to adjust the threshold value τ_p *locally* for each inverted list $I(t_i)$, so that we maintain *at least a certain number of postings for each inverted list $I(t_i)$* . This policy is shown on Figure 5.8. Unfortunately, it is not possible to compute the correctness function $\mathcal{C}$ just from the p-index constructed this way.

Theorem 6 *No PR-based vertical pruning can provide the result guarantee.* □

Proof Assume that we create a p-index by using any of the above vertical pruning algorithms. Consider the minimum pr_{min} value from all the postings that ended up in the p-index. Now consider a query q that is answered using the p-index and that the last (k^{th}) document in the answer has a ranking score of $r(D_k, q) = f_r(tr(D_k, q), pr(D_k))$. Now consider a document D_j that is not in the p-index because it was pruned, i.e.

$pr(D_j) < pr_{min}$. This document's rank is $r(D_j, q) = f_r(tr(D_j), q, pr(D_j))$. Although we know that $pr(D_j) < pr_{D_k}$, we have no way of knowing the relationship between $tr(D_k)$ and $tr(D_j)$ and therefore we cannot guarantee that D_k should be in the final answer. ■

Therefore, under this pruning policy, the quality of the answer computed from the p-index can be significantly worse than that of the full answer and it is not possible to detect this degradation without computing the answer from the full index. In the next section, we propose simple yet essential changes to this pruning policy that allows us to compute the correctness function $\mathcal{C}$ from the p-index itself.

5.5.2.2 Extended keyword-specific pruning

As Theorem 6 demonstrated, by using one of the previous pruning policies, the relationship between the query-dependent relevance of the documents in the p-index and the ones in the full index is unknown. To alleviate this problem, we propose a vertical pruning policy that we call *extended keyword-specific pruning*, and which prunes based on both the $pr(D_i)$ and $tr(D_i, t_i)$ values.

At a high level, this vertical pruning policy operates as follows. First, we determine the number N which is the maximum size of an individual posting list. Then, for every posting list associated with keyword t_i , we keep everything if the size of the list is smaller than N . If the size is larger than N , we pick two thresholds τ_p and τ_r and keep the postings D_i if $pr(D_i) > \tau_p$ or $tr(D_i, t_i) > \tau_r$. This process is shown on Figure 5.9.

The thresholds τ_p and τ_r may be picked in a number of different ways depending on the exact ranking function. For example assuming that pr and tr have equal weight in the final ranking function, we may pick the τ_p and τ_r as follows: Conceptually, we generate two sorted posting lists for each keyword t_i , one based on $pr(D_i)$ and the

Algorithm 5.5.5 *Extended keyword-specific vertical pruning* VS_E **Input** N : maximum size of a single posting list**Procedure**

- (1) N_i : total number of documents in the list i
- (2) If $N \geq N_i$ keep all D_i
- (3) Else:
- (4) pick thresholds τ_p and τ_r .
- (5) keep D_i if $pr(D_i) > \tau_p$ or $tr(D_i, t_i) > \tau_r$.

Figure 5.9: Extended keyword-specific vertical pruning based on pr and tr .

other based on $tr(D_i, t_i)$. Then we pick a number m such that the union of the top m documents in $pr(D_i)$ sorted list and in the $tr(D_i, t_i)$ sorted list have N documents. Then the lowest $pr(D_i)$ and $tr(D_i, t_i)$ scores in the union are the threshold values τ_p and τ_r respectively.

Given the fact that we prune away a number of postings from the inverted index, it may be the case that, for some of the queries, the p-index will not be able to calculate the same top- k results as I_F . Since one of our requirements is to provide a guarantee for the cases that we cannot correctly compute the result, we will now describe how we can to provide this guarantee within a vertically pruned index.

Formally, consider a vertically pruned index I_P and a query $q = \{t_1, \dots, t_w\}$. Our goal is to determine whether we can correctly calculate the top- k results by using I_P . We assume that the results are computed based on the AND semantics that we described in Section 5.2.2 and that a monotonic ranking function (such as Equation 5.4) is used for the ranking of documents.

For every term in q we calculate the lowest values $tr_\tau(t_1), \dots, tr_\tau(t_n)$ (we will call

Algorithm 5.5.6 *Ranking based on thresholds $tr_\tau(t_i)$ and $pr_\tau(t_i)$*

Input Query $q = \{t_1, \dots, t_w\}$

Procedure

- (1) $\forall D_i \in S = I_P(t_1) \cup \dots \cup I_P(t_w)$
// lists that D_i appears in
- (2) Find $J = \{I_P(t_m) : D_i \in I_P(t_m)\}$
// lists that D_i does not appear in
- (3) Find $R = \{I_P(t_m) : D_i \notin I_P(t_m)\}$
- (4) If $R = \emptyset$
- (5) return $r(D_i, q) = f_r(tr(D_i, q), pr(D_i))$
- (6) Else
- (7) $tr^*(D_i, q) = f_{tr}(tr^*(D_i, t_1), \dots, tr^*(D_i, t_n))$
- (8) with $tr^*(D_i, t_m) = \begin{cases} tr(D_i, t_m) & \text{if } I_P(t_m) \in J \\ tr_\tau(t_m) & \text{if } I_P(t_m) \in R \end{cases}$
- (9) return $r(D_i, q) = f_r(tr^*(D_i, q), pr(D_i))$

Figure 5.10: Ranking based on thresholds $tr_\tau(t_i)$ and $pr_\tau(t_i)$.

these values threshold values) in the respective inverted list within I_P . In addition, since we keep the pr values also in the posting lists, we also calculate the lowest pr values for the documents in every list: $pr_\tau(D_1), \dots, pr_\tau(D_n)$.

In order to determine the result for the query q , for every document D in the list of documents we calculate its rank $r(D, q)$ as follows: If the document appears in all of the inverted lists, this means that none of its postings has been vertically pruned. In this case, we can use the $tr(D, t_i)$ values and the document's $pr(D)$ to calculate its rank according to Equation 5.4: $r(D, q) = f_r(tr(D, q), q(D))$. If the document does not appear in all of the inverted lists, we calculate the upper bound of its ranking by using the threshold values for the missing lists.

Example 5.5.1 Assume that $q = \{t_1, t_2\}$ and that D appears in the lists of pr, t_1 but not in the list of t_2 . Because the index has been vertically pruned, we don't know whether the absence of D from the list of t_2 is because D does not contain t_2 or because its posting has been vertically pruned. For this reason we will use the threshold value $tr_\tau(t_2)$ in the ranking function of Equation 5.4. For example, if we assume that $r(D, q) = \alpha \cdot tr(D, q) + (1 - \alpha) \cdot pr(D)$ and that $tr(D, q) = tr(D, t_1) + tr(D, t_2)$, then we can calculate the thresholded rank $r_\tau(D, q)$ as follows: $r_\tau(D, q) = \alpha \cdot [tr(D, t_1) + tr_\tau(t_2)] + (1 - \alpha) \cdot pr(D)$.

The formal algorithm for ranking based on the two thresholds $tr_\tau(t_i)$ and $pr_\tau(t_i)$ is shown in Figure 5.10. Once we use the above way of ranking within the p-index, we can provide a guarantee that the top- k computed results are the same as the full index, based on the following theorem:

Theorem 7 Given a vertically pruned inverted index I_P , a query $q = \{t_1, \dots, t_n\}$ and a monotonic ranking function operating according to Algorithm 5.10, we are guaranteed that the top- k results of I_P are the same as the top- k of the full index I_F , if none of the top- k results used a threshold value for its ranking. $\square$

Proof For simplicity, let us assume that $k = 3$ and that the result list contains documents $D_1, D_2, D_3, D_4, D_5, \dots, D_m$. For every document D_i in I_F there are three possible scenarios regarding the final answer:

1. D_i is in the answer because it appears in all inverted lists. In this case we can calculate its exact rank according to Algorithm 5.10.
2. D_i is not in the final answer because it was pruned from all lists. In this case, we know that for D_i it holds that $pr(D_i) < \min_{1 \leq j \leq n} p_\tau(t_j)$ and that $tr(D_i, t_j) < tr_\tau(t_j), 1 \leq j \leq n$. From the monotonicity assumption it follows that $r(D_i) \leq r(D_m)$, i.e. every pruned D_i cannot be ranked higher than the last document D_m and thus it cannot belong in the answer.
3. D_i is in the answer because a threshold value was used for its ranking. In this case we can determine whether the answer is correct as follows. Assume that for the ranking of D_2 a threshold value was used. Because the ranking function is monotonic, it follows that $r_\tau(D_2, q) \geq r(D_2, q)$. That is, the thresholded rank of D_2 is at least as much as its real rank computed from the full index. However, without knowing the real rank of D_2 , we have no way of knowing how close the r_τ and r values are. Therefore, we cannot guarantee that the position of D_2 in the ranked list is the correct one.

Similarly, let us assume that for the ranking of D_4 a threshold value was used. Due to the monotonicity of the ranking function, it follows that $r_\tau(D_4, q) \geq r_\tau(D_i, q)$ for $i > 4$, and that $r_\tau(D_4, q) \geq r(D_i, q)$ for $i > 4$. Remember however, that $r_\tau(D_4, q)$ is an upper bound of $r(D_i, q)$. That is the rank value of D_4 based on the full index can be at most $r_\tau(D_4, q)$, and the same holds for the rest of the documents after D_4 . Therefore the position of D_4 in the full index can be at most where it is by using a threshold value. Since no threshold value was used

for D_1, D_2, D_3 we are certain that their order will be the same as the one from the full index. Therefore, in this case we can guarantee that the computed result is correct. ■

In Section 5.7 we will present experimental evaluation of our pruning policies.

5.6 Index maintenance

In a practical scenario search engines need to periodically update their indexes in order to cope with the changing information. The index updates are necessary mainly due to two reasons: First, the *Web pages change* over time and the search engines need to refresh their collections in order to provide users with up-to-date information. Second, the overall *queries* received by a search engine *change* over time. In the case where we use a pruning technique that operates based on the query load, we may have to update our index to accommodate for new kinds of query loads.

In order to cope with the *changes on the Web* the search engines regularly redownload the Web pages and rebuild their indexes. In most cases, in order not to miss any important changes on the Web they redownload pages that are likely to change more often than others (e.g. they may redownload news Web sites more often than university pages). In this case, whenever we rebuild the full index, we can also rebuild the p-indexes by using the algorithms provided in Section 5.5.

In the case where the *query load changes* over time, we may need to update our p-index to accommodate for this change. More specifically, during the horizontal pruning algorithm, we assume that we are given the query load that will be used on our index. In a practical scenario, we may not have this information. However, we may be able to use previously seen query loads as an approximation for the future ones. For example, we may be able to use the query load of the previous month as an ap-

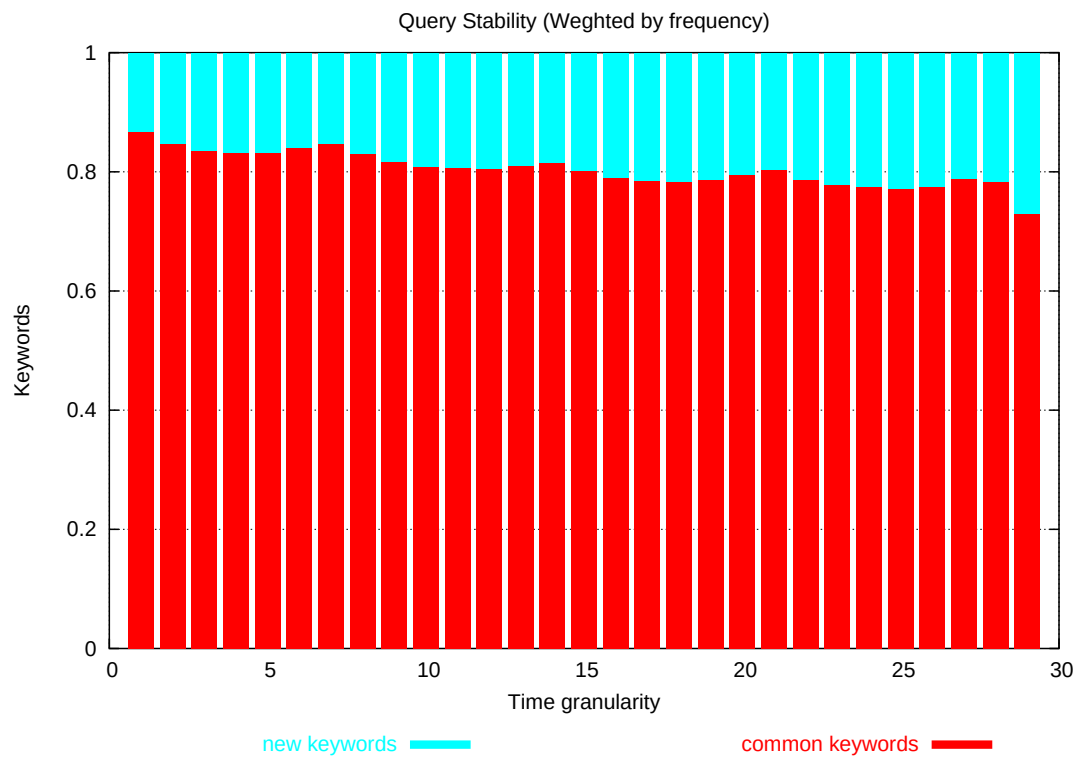


Figure 5.11: Query-load stability over time.

proximation of the query load that we expect to observe during the current month. If these two query loads are very similar, then our horizontal pruning will perform well; if the query loads are different, then we may observe a degradation in the number of guaranteed queries.

In order to study this problem, we performed an experiment studying the query stability, which we show in Figure 5.11. Figure 5.11 shows the fraction of keywords in each day of the query load that are *common* with the keywords that had appeared in the first day of the query load. The graph is obtained from the daily query load on one of the commercial search engines. We will give a more detailed description of this dataset in Section 5.7.

The horizontal axis corresponds to the day and the red bar indicates how many of keywords in each day are common with the keywords issued on the first day (weighted by their frequencies). For example, the fifth bar of the graph shows that about 82% of the keywords issued on the fifth day appeared in the queries on the first day. The graph shows that the queries on every day share a large number of keywords over a relatively long period, with 80% of the keywords commonly appearing over one month. We also conducted a number of other experiments measuring the stability of the query load and obtained similar results indicating that the query load is relatively stable. Based on these results, we assume that we can use the query load of the recent past (say, the last month), as the expected query load of the near future (say, the coming month).

5.7 Experimental Evaluation

5.7.1 Data and experimental setup

In this section we experimentally evaluate the pruning policies presented in this chapter. Our goal is to validate our theoretical analysis through real-world experiments, by

creating a p-index according to the algorithms proposed in Section 5.5.

In order to perform realistic tests for our pruning policies, we implemented a search engine prototype, which will be described in more detail in Chapter 7. For the experiments in this chapter, our search engine indexed about 130 million pages, crawled from the Web during March of 2004. The crawl started from the Open Directory’s [Dmo] homepage and proceeded in a breadth-first manner. Najork and Wiener [NW01] showed that crawling in this way yields well-connected pages of high “quality” and high PageRank [PBM99] values. For every page within our collection, we calculated its PageRank value and we stored it in a separate database that allows efficient random accesses. Overall, the total uncompressed size of our crawled Web pages is approximately 1.9 TB, yielding a full inverted index I_F of approximately 1.2 TB.

For the experiments reported in this section we used a real set of queries issued to Looksmart [Loo] on a daily basis during April of 2003. After keeping only the queries containing keywords that were present in our inverted index, we were left with a set of about 462 million queries. Within our query set, the average number of terms per query is 2 and 98% of the queries contain at most 5 terms.

5.7.2 Horizontal pruning

In our first experiment we study the performance of the horizontal pruning, described in Section 5.5.1. More specifically, we apply the algorithm HS of Figure 5.5 to our full index I_F and create a *horizontally-pruned* p-index I_P of size s . For the construction of our horizontally pruned p-index we used the query frequencies observed during the first 10 days of our data set. Then, we calculated the fraction $f(s)$ of the rest of the queries (i.e. the queries observed in the following 20 days) that are *guaranteed to be correct* by I_P . According to the algorithm of Figure 5.6, a query can be guaranteed to

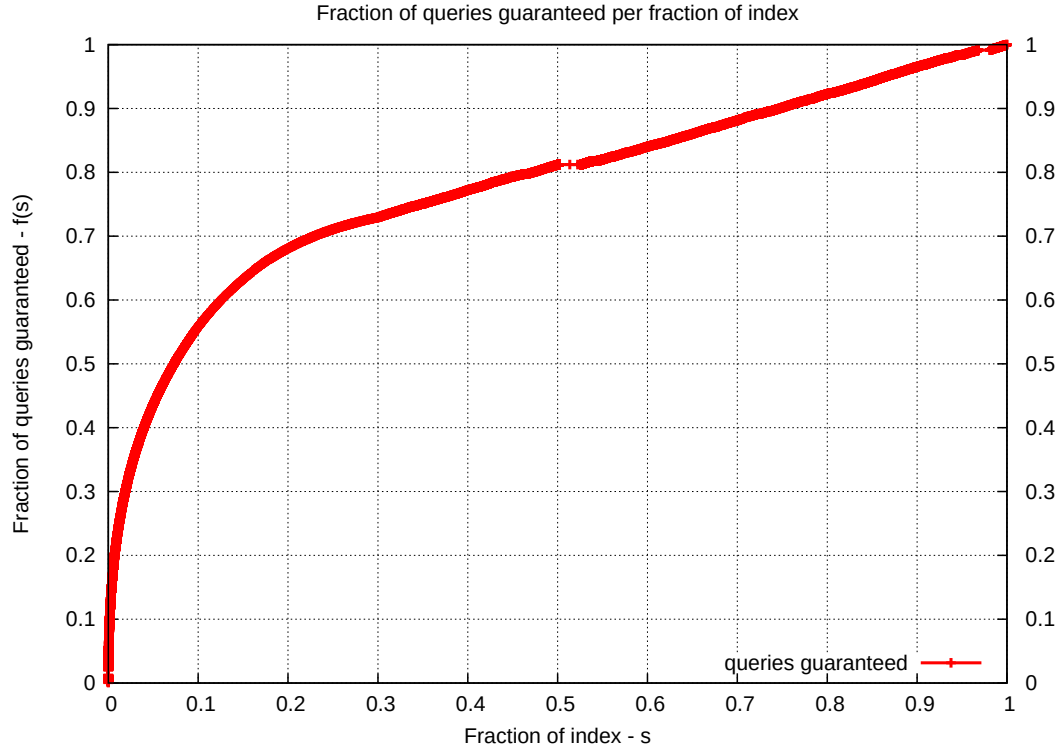


Figure 5.12: Fraction of *guaranteed queries* $f(s)$ answered in a horizontally-pruned p-index of size s .

be correct, if I_P includes the inverted lists for *all* of the query's keywords.

We have repeated the experiment for varying values of s , picking the keywords greedily as discussed in Section 5.5.1. The result is shown in Figure 5.12. The horizontal axis denotes the size s of the p-index as a fraction of the size of I_F . The vertical axis shows the fraction $f(s)$ of the queries that the p-index of size s can answer.

The results of Figure 5.12, are very encouraging: we can answer a significant fraction of the queries with a small fraction of the original index. For example, approximately 73% of the queries can be answered using 30% of the original index.

Since the performance of the horizontal pruning depends on the query load used for the pruning, we also studied how well it performs over time. For this reason, we again

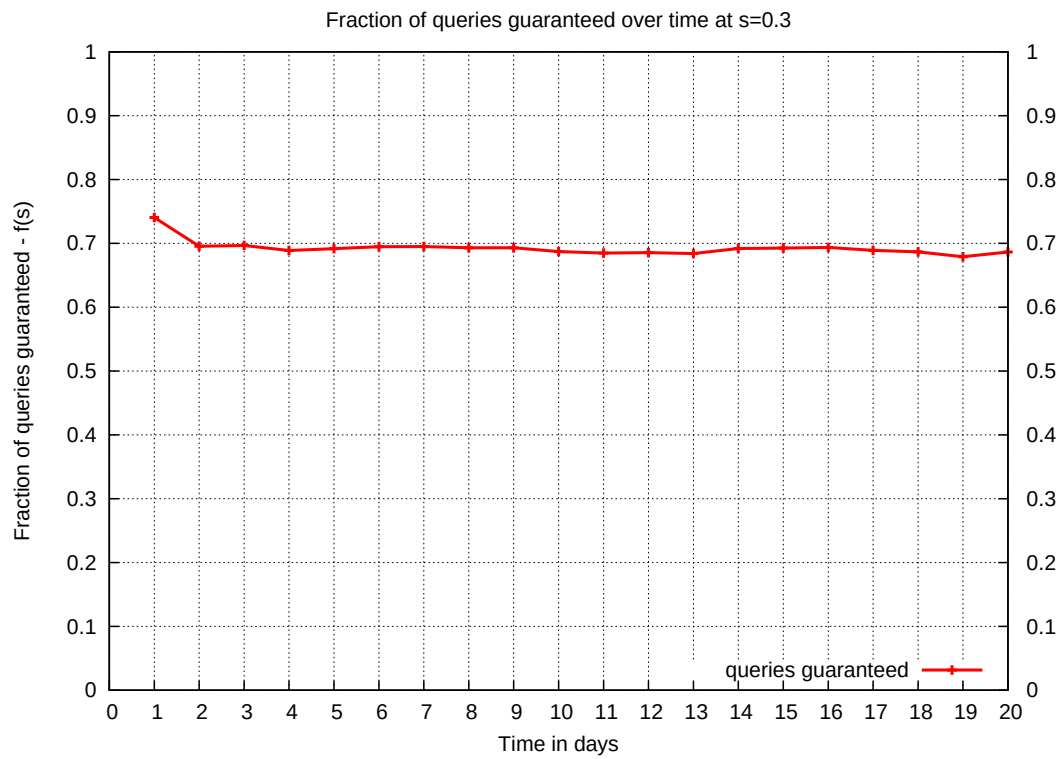


Figure 5.13: Fraction of *guaranteed queries* $f(s)$ answered in a horizontally-pruned p-index of size $s = 0.3$ over time.

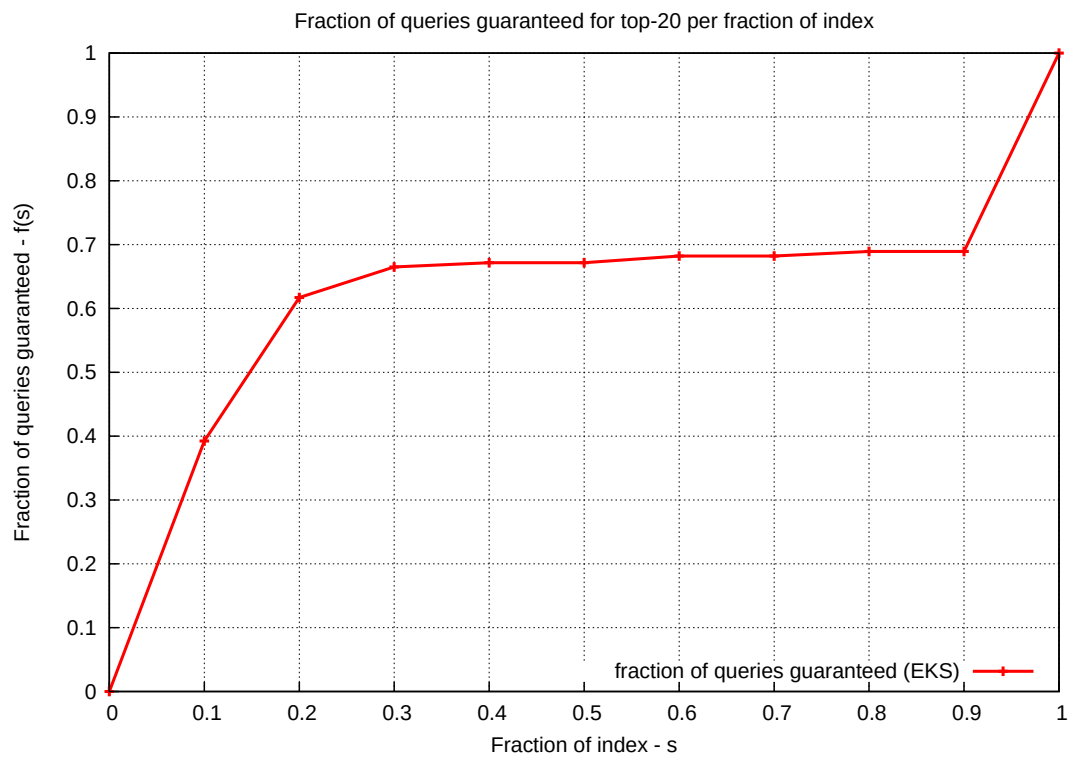


Figure 5.14: Fraction of *guaranteed* queries $f(s)$ answered in a vertically-pruned p-index of size s .

generated a horizontally pruned index based on the first 10 days of our query load, and we calculated the fraction of guaranteed queries for each one of the remaining 20 days. We present the results through the graph shown in Figure 5.13. The horizontal axis depicts the time in days and the vertical axis shows the fraction of guaranteed queries in each day. The graph shown in Figure 5.13 corresponds to an index size of $s = 0.3$ (the observations that we obtained for different index sizes are analogous).

Figure 5.13 shows that the performance of our horizontally pruned index remains relatively stable over time. In other words, given our horizontally-pruned index, we expect it to be able to guarantee a large fraction of queries (about 70% in our case) several days after first generating our p-index.

We now proceed to examine the benefits of the vertical pruning policies.

5.7.3 Vertical pruning

We continue our experimental evaluation by studying the performance of the various vertical pruning policies described in Section 5.5.2. For the experiments on vertical pruning reported here we worked with a 5.5% sample of the whole query set. The reason behind this is merely practical: since we have much less machines compared to a commercial search engine it would take us about a year of computation to process all 462 million queries.

Within an inverted index (p-index or full), for every query $q = \{t_1, \dots, t_w\}$, we calculated the rank of a matching document D by weighting the document quality and the query-dependent relevance as follows:

$$r(D, q) = pr_{norm}(D) + tr_{norm}(D, q) \quad (5.5)$$

where $pr_{norm}(D)$ is the normalized PageRank of D :

$$pr_{norm}(D) = \frac{\log(pr(D)) - \log(pr_{min})}{\log(pr_{max}) - \log(pr_{min})} \quad (5.6)$$

with pr_{min} and pr_{max} being, respectively, the minimum and maximum PageRank values within the set of documents matching q . Since there exists orders of magnitude difference in pr values, it is common to take the log of pr in the ranking function [LS03].

Similarly, we calculate the normalized query-dependent relevance:

$$tr_{norm}(D, q) = \frac{tr(D, q) - tr_{min}}{tr_{max} - tr_{min}} \quad (5.7)$$

with tr_{min} and tr_{max} being, respectively, the minimum and maximum query - dependent relevance within the set of documents matching q . The value $tr(D, q)$ is calculated by using the TF.IDF cosine distance of Equations 5.1 and 5.2. We should note that we did not engineer the ranking function in any way. The above rankings are similar to the ranking function employed in [LS03].

The results presented in this section depend on the choice of the exact ranking functions. For example if there is very small weight given to the PageRank then our global and local pruning schemes may not perform well. However, based on the services that Search Engine Optimization (SEO) firms offer on the Web, it appears that both term relevance and PageRank are important in the real Web.

For our first experiment, we generate a *vertically-pruned* p-index of size s by using the *Extended Keyword-Specific pruning* (henceforth referred to as *EKS*), described in Section 5.5. Within the p-index we measure the fraction of queries that can be guaranteed (according to Theorem 7) to be correct. We have performed the experiment for varying index sizes s and the result is shown in Figure 5.14. Based on this figure, we

can see that our vertical pruning algorithm performs well across the scale of index sizes s : for all index sizes larger than 40%, we can guarantee the correct answer for about 70% of the queries. This implies that our *EKS* algorithm can successfully identify the necessary postings for calculating the top-20 results for 70% of the queries by using at least 40% of the full index size.

We can compare the two pruning schemes, namely the horizontal pruning and *EKS*, by contrasting Figures 5.12 and 5.14. Our observation is that, if we would have to pick one of the two pruning policies (we discuss the combination of the two policies in Section 5.7.5), then the two policies seem to be more or less equivalent for the p-index sizes $s \leq 20\%$. For the p-index sizes $s > 20\%$, horizontal pruning does a much better job as it provides a higher number of guarantees at any given index size.

In our next experiment, we are interested in comparing *EKS* with the other two vertical pruning policies that we proposed in Section 5.5.2. To this end, apart from *EKS*, we also generated *vertically-pruned* p-indexes for the *Global pr-based pruning* (henceforth referred to as *GPR*), and the *Local pr-based pruning* (henceforth referred to as *LPR*) policies. For each of the policies we created vertically-pruned p-indexes of varying sizes s . Since *GPR* and *LPR* cannot provide a correctness guarantee, we will compare the fraction of queries from each policy that are *identical* (i.e. the same results in the same order) to the top- k results calculated from the full index. Here, we will report our results for $k = 20$; the results are similar for other values of k . The results are shown in Figure 5.15. The horizontal axis shows the size s of the p-index; the vertical axis shows the fraction $f(s)$ of the queries whose top-20 results are identical to the top-20 results of the full index, for a given size s .

By observing Figure 5.15, we can see that *GPR* performs the worst of the three policies. On the other hand *EKS*, picks up early, by answering a great fraction of queries (about 62%) correctly with only 10% of the index size. The fraction of queries

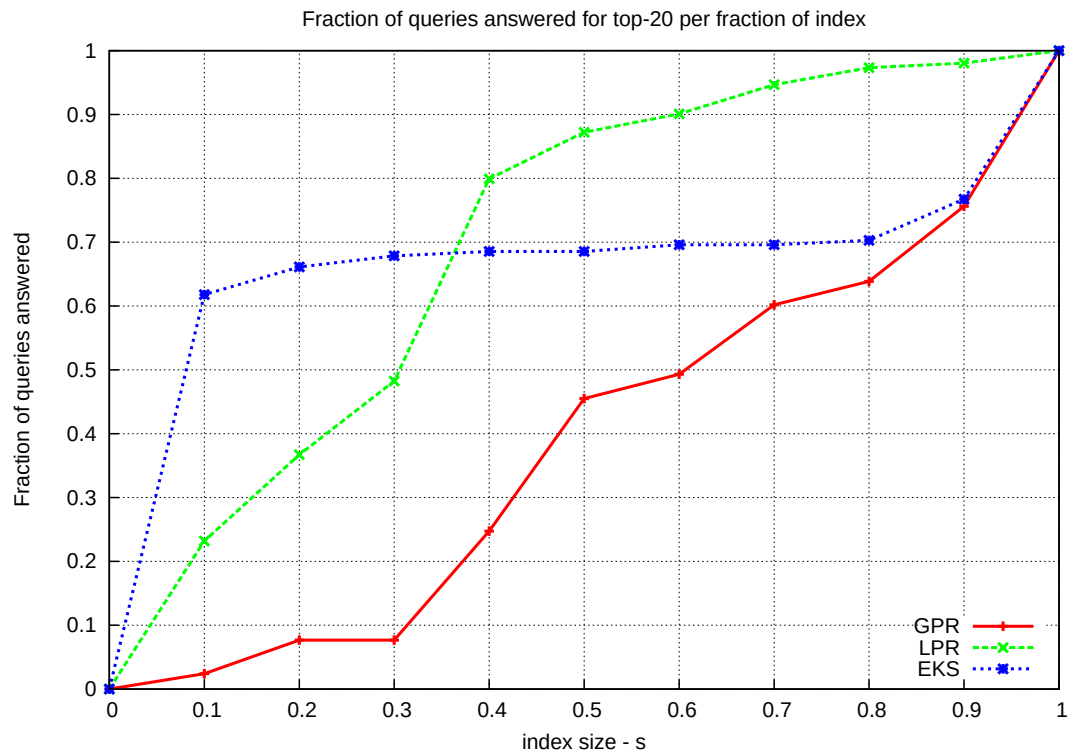


Figure 5.15: Fraction of queries answered in a vertically-pruned p-index of size s .

that *LPR* can answer remains below that of *EKS* until about $s = 37\%$. For any index size larger than 37%, *LPR* performs the best.

In the experiment of Figure 5.15, we applied the strict definition that the results of the p-index have to be in the same order as the ones of the full index. However, in a practical scenario, it may be acceptable to have some of the results out of order. Therefore, in our next experiment we will measure the fraction of the results coming from an p-index that are contained within the results of the full index. The result of the experiment is shown on Figure 5.16. The horizontal axis is, again, the size s of the p-index; the vertical axis shows the average fraction of the top-20 results common with the top-20 results from the full index. Overall, Figure 5.16 depicts that *EKS* and *LPR* identify the same high ($\approx 96\%$) fraction of results on average for any size $s \geq 30\%$, with *GPR* not too far behind.

5.7.4 Optimal p-index size

In Section 5.4 we performed a theoretical analysis on how to calculate the optimal size s of an p-index, which minimizes our query cost. In this section, we show how to experimentally calculate s for the horizontal pruning policy (the calculation for the vertical pruning can be performed in a similar fashion).

Given the function $f(s)$ of Figure 5.12, we can now calculate the optimal size s by using our analysis in Section 5.4 as follows: First, we get an analytical form of the function $f(s)$ of Figure 5.12, by performing fitting on the data points. We found that $f(s)$ can be approximated well by the polynomial:

$$f(s) = 0.135 + 6.779 \cdot s - 29.432 \cdot s^2 + 60.730 \cdot s^3 - 57.567 \cdot s^4 + 20.401 \cdot s^5$$

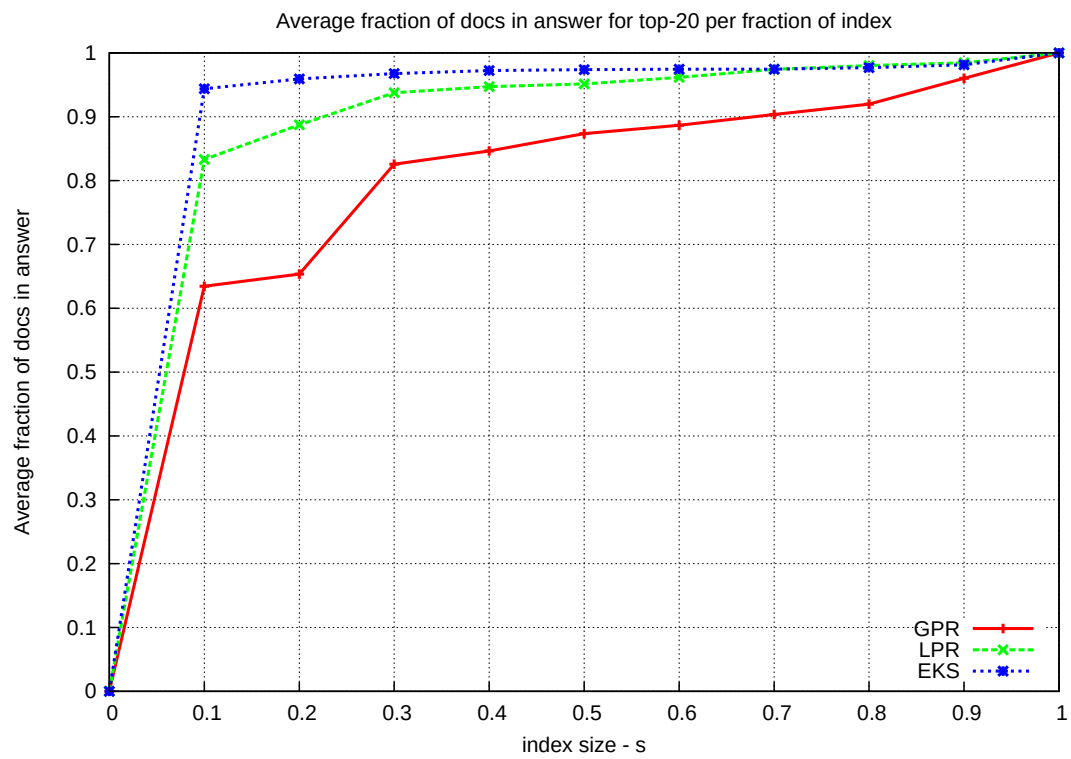


Figure 5.16: Average fraction of the top-20 results of p-index with size s contained in top-20 results of the full index.

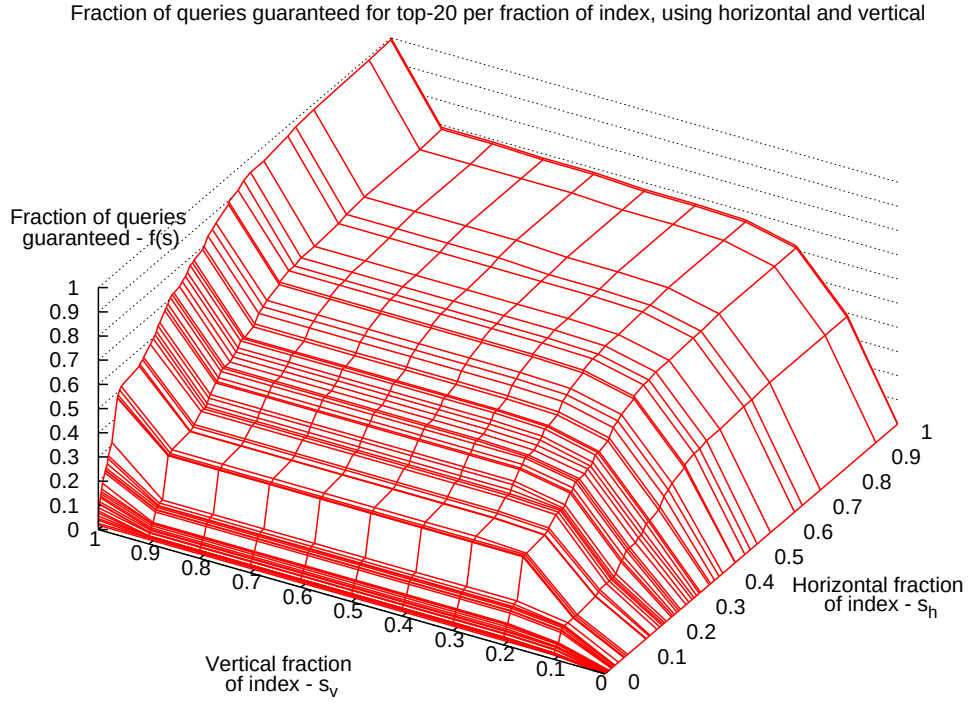


Figure 5.17: Combining horizontal and vertical pruning.

By applying Theorem 4, we can now calculate the optimal index size as $s = 0.179$. That is, we can minimize our query cost when we create a horizontally-pruned p-index of about 17.9% of the full index's size.

5.7.5 Combining horizontal and vertical pruning

In Sections 5.7.2 and 5.7.3 we studied the individual performance of our horizontal and vertical pruning schemes. One interesting question however is how do these policies perform in combination? What fraction of queries can we guarantee if we apply both horizontal and vertical pruning in our full index I_F ?

To answer this question, we performed the following experiment. We started with the full index I_F and we applied horizontal pruning to create an index I_P^h of size s_h .

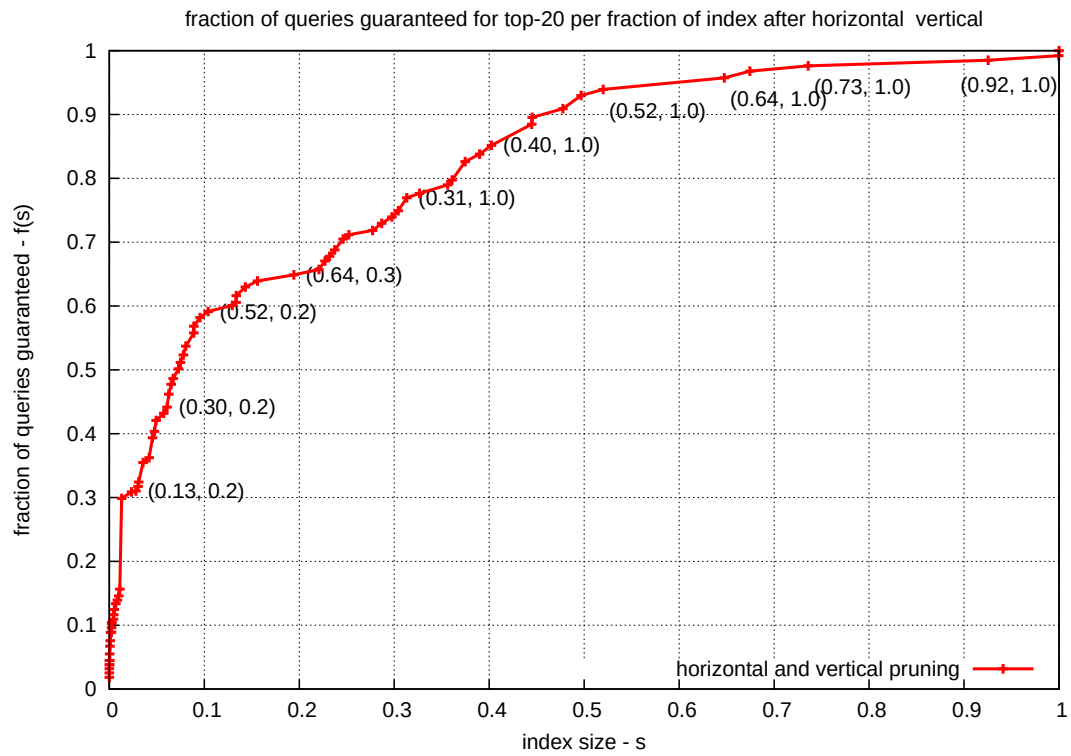


Figure 5.18: Performance of combined horizontal and vertical pruning.

100% of I_F . After that, we further applied vertical pruning to I_P^h , and created our final p-index I_P of size $s_v \cdot 100\%$ of I_P^h . We then calculated the fraction of guaranteed queries in I_P . We repeated the experiment for different values of s_h and s_v . The result is shown on Figure 5.17. The x-axis shows the index size s_h after applying horizontal pruning; the y-axis shows the index size s_v after applying vertical pruning; the z-axis shows the fraction of guaranteed queries after the two prunings. For example the point (0.2, 0.3, 0.4) means that if we apply horizontal pruning and keep 20% of I_F , and subsequently on the resulting index we apply vertical pruning keeping 30% (thus creating a p-index of size $20\% \cdot 30\% = 6\%$ of I_F) we will be able to guarantee 40% of the queries.

By observing Figure 5.17, we can see that for p-index sizes smaller than 50%, our combined pruning does relatively well. For example, by performing 40% horizontal and 40% vertical pruning (which translates to a pruned index with $s = 0.16$) we can provide a guarantee for about 60% of the queries. In Figure 5.17, we also observe a “plateau” for $s_h > 0.5$ and $s_v > 0.5$.

In order to investigate the combination of horizontal and vertical pruning in more detail, we converted the graph of Figure 5.17 to a 2-dimensional version as shown in Figure 5.18. The horizontal axis shows the eventual size s of our p-index after the two prunings; this is essentially a multiplication of a pair of s_h and s_v values from Figure 5.17. The vertical axis shows the fraction of queries that we can guarantee to be correct at a given size s . In the parentheses we denote the (s_h, s_v) values that correspond to a given point in the graph. If two or more pairs of s_h and s_v from Figure 5.17 would yield the same s , we kept the pair with the highest guarantee. For example, for $s \approx 0.1$, we can guarantee about $f(s) = 60\%$ of the queries, if we apply horizontal pruning with $s_h = 0.52$ and then vertical pruning with $s_v = 0.2$. The graph shows that for p-index sizes $s < 0.3$, we get the highest fraction of guaranteed queries when we employ both horizontal and vertical pruning. In this range, vertical pruning

typically creates a p-index with size s_v between 20 – 30%. For values of $s > 0.3$, horizontal pruning takes over; for this range of p-index sizes we can get the highest fraction of guaranteed queries by simply performing horizontal pruning.

5.8 Related Work

For an overview of *inverted indexing* in Web search engines and IR systems in general, the reader may refer to [ACG01, BR99, WMB99]. Experimental studies and analyses of various *partitioning schemes* for an inverted index are presented in [CML00, JO95, MRY01, TG93]. The pruning algorithms that we have presented in this dissertation are independent of the partitioning scheme used.

The exact *ranking functions* that are being used by current search engines are closely guarded secrets. It is common knowledge, however, that the results are ranked mainly based on a query-dependent relevance and a query -independent document “quality” (as we discussed in Section 5.2). Query-dependent relevance can be calculated in a variety of ways (see [BR99, SM83, WMB99]). Similarly, there are a number of works that measure the “quality” of the documents, typically as captured through link-based analysis [Hav02, Kle99, LM00, RD02, PBM99]. Our work is independent of the exact ranking function that is being used; our only requirement is the ranking function to be monotonic.

There has been a great body of work on *top-k result calculation* in the Databases and IR community. The main idea is to either stop the traversal of the inverted lists early, or to shrink the lists by pruning postings from the lists [BL85, Fag, Fag99, WL93]. The works in [AKM01, CCF01, LS03, PZS96], are the most related to ours, as they describe pruning techniques based on the idea of keeping the postings that contribute the most in the final ranking. However, [AKM01, CCF01, PZS96] do not con-

sider any query-independent quality (such as PageRank) in the ranking function. Long and Suel, do investigate pruning algorithms that involve PageRank, in a work [LS03] mainly geared towards optimizing query throughput and response time. The works in [BGM02, CG96, GBK01], present algorithms for determining the top- k results in (mainly multimedia) databases under different scenarios. For example [BGM02] considers the case where (some of) the data are not sorted, while [CG96] devises efficient query plans within a relational database. Our work essentially extends that of [AKM01, BGM02, CCF01, LS03, PZS96], by requiring our index to provide a *correctness indication* along with the result, and by proposing modifications to the pruning techniques that make this possible. Our approach on the correctness indication was inspired by [FLN01].

Search engines typically use various methods of *caching* as a means of reducing the cost associated with every query. With result caching, we can take advantage of repeated queries in the query stream by caching the computed result [LM03, LM04, Mar01, XO02]. Similarly, but operating at a lower level, we can cache frequently-accessed inverted lists [LS05, SMZ01]. There have also been proposed multi-level caching schemes where we can cache both lists and results [SMZ01], or we can cache lists, results and projection lists (i.e. pairs of frequent terms) [LS05] to improve query response times. Determining which parts of an index to cache is, in principle, similar to pruning an index to keep the most important postings. In this dissertation, we showed how a pruned index can provide us with a *correctness indication* of the results. The methods and algorithms that we presented can potentially be used to extend existing caching schemes. The exact details can be an interesting avenue for future work.

As a final note, we should remark that various pruning or caching techniques are most probably employed by current search engines. Since this kind of information is kept confidential, we are not aware of the techniques applied in a real search engine.

5.9 Summary

Web search engines typically prune their large-scale inverted indexes in order to scale to enormous query loads. While this approach may improve performance, by computing the top results from a pruned index we may notice a significant degradation in the result quality.

In this chapter, we studied how we can avoid the degradation of search quality. Our algorithms were explored in the context of the popular *cluster architecture* that search engines employ today. In particular, we studied new pruning techniques and answer computation algorithms that *guarantee* that the top matching pages are *always* placed at the top of search results, although we use a pruned index. We provided theoretical justification of our proposed techniques and experimental evaluation on real Web data. Our results showed that both our horizontal and vertical pruning schemes can effectively be used to to prune an inverted index without any degradation in the quality of the results.

CHAPTER 6

Content-based Spam Detection

6.1 Introduction

From its modest beginnings as a means to share Physics data, the Web has grown to be a central part of cultural, educational and, most importantly, commercial life. Millions of users today are performing financial transactions on Web pages, varying from buying goods, to booking travel and hotels, to applying for credit cards or mortgages. Due to the astonishing amount of information available on the Web, users typically locate useful Web pages by querying a search engine. Given a query, a search engine identifies the relevant pages on the Web and presents the users with the links to such pages, typically in batches of 10–20 links. Once the users see relevant links, they may click on one or more links in order to visit the pages.

In the last few years, this model of reaching relevant information through the use of search engines has become pervasive. Several sites on the Web observe an ever-increasing portion of their traffic coming from search engines referrals. For example, Jacob Nielsen’s site HypertextNow, which attracts visitors interested in Web ratings and usability, receives about a third of its traffic through such referrals [Nie], and search engine referrals have increased by about 10% in the last year.

For many commercial Web sites, an increase in search engine referrals translates to an increase in sales, revenue, and, one hopes, profits. According to the US Census Bureau, total US e-Commerce sales in 2004 amounted to \$69.2 billion (or 1.9% of

total US sales), and Web-based e-Commerce continues to grow at a rate of 7.8% per year [Cen]. Forrester Research predicts that online US business-to-consumer sales of goods including auctions and travel will amount to \$172 billion in 2005 [Joh], and will grow to \$329 billion in 2010, accounting for 13% of all US retail sales. In order for commercial Web sites to tap into this accelerating market, they have to increase their traffic, which is tantamount to showing up in the first few¹ search engine results.

Given the large fraction of Web traffic originating from searches and the high potential monetary value of this traffic, it is not surprising that some Web site operators try to influence the positioning of their pages within search results. Some operators attempt to influence their positioning through ethical, or white-hat, Search Engine Optimization (SEO) techniques, improving the quality and appearance of their content and serving content useful to many users.

However, many Web site operators try to manipulate the ranking functions of search engines by using less-ethical gray-hat and black-hat SEO techniques. These include the creation of extraneous pages which link to a target page (link stuffing). Using link stuffing, Web sites can increase the desirability of target pages to search engines using link-based ranking. The content of other pages may be engineered so as to appear relevant to popular searches, a technique known as keyword-stuffing. The hope is that the target pages will rank high within the search engine results for included terms and thereby draw users to visit their Web sites. The practices of crafting Web pages for the sole purpose of increasing the ranking of these or some affiliated pages, without improving the utility to the viewer, are called Web spam. Figure 6.1 shows an example of a spam page: this page contains important keywords, however its content is, on the whole, useless to a human viewer. In the context of search engines, spam

¹A recent study [JS03] showed that approximately 80% of search engine users look at no more than the first 3 batches of results. Therefore, unless a site is listed in the first few results, it is highly unlikely to see its traffic increasing.

can be a great nuisance for several reasons. First, since there are financial advantages to be gained from search engine referrals, Web site operators generating spam deprive legitimate sites of the revenue that they might earn in the absence of spam. To the extent that search engines allow spam to distort recommendations, they are unwittingly complicit in this unfairness towards legitimate sites. Second, if a search engine returns spam Web pages to its users, they will be confronted with irrelevant results, which may lead to frustration and disappointment in the search engines services. Finally, a search engine may waste significant resources on spam pages. Unless detected, the spam pages are crawled (thus wasting network bandwidth), processed (thus wasting CPU cycles), indexed (thus wasting storage space), and matched against queries (wasting CPU cycles and disk bandwidth in matching, and network bandwidth when returning results). Given the amount of spam on the Web (estimated to be 13.8% of English-language pages, as shown in Section 6.3), a search engine which does not distinguish spam wastes one-seventh of its resources for spam alone.

Creating an effective spam detection method is a challenging problem. Given the size of the Web, such a method has to be automated. However, while detecting spam, we have to ensure that we identify spam pages alone, and that we do not mistakenly consider legitimate pages to be spam. At the same time, it is most useful if we can detect that a page is spam as early as possible, and certainly prior to query processing. In this way, we can allocate our crawling, processing, and indexing efforts to non-spam pages, thus making more efficient use of our resources.

In this dissertation, we explore a variety of methods for detecting spam. Each method is highly parallelizable, can run in time proportional to the size of the page, and identifies spam pages by analyzing the content of every downloaded page. We present experiments performed on a subset of a crawl performed by MSN Search [MSN] demonstrating the relative merits of every method. We also present how to employ

machine learning techniques that combine our individual methods to create a highly efficient and reasonably-accurate spam detection algorithm. The approaches described in this dissertation extend previous work in identifying Web spam [FMN04, FMN05].

The remainder of this chapter is structured as follows: In Section 6.2 we describe our experimental framework and the real-world data set that we used. In Section 6.3 we estimate the prevalence of spam in our data set. In Section 6.4 we describe the spam-detection methods that we explored and in Section 6.5 we examine how well these methods can work in combination. In Section 6.6 we discuss related work, and, finally, in Section 6.7 we offer some concluding remarks and outline directions for future work.

6.2 Experimental framework and data set

In order to design and evaluate our spam detection algorithms, we used a collection of 105, 484, 446 Web pages, collected by the MSN Search [MSN] crawler, to serve as a proxy for the Web at large. These pages were collected during August 2004, and were drawn arbitrarily from the full MSN Search crawl.

The MSN Search crawler discovers new pages using a roughly breadth-first exploration policy, and uses various importance estimates to schedule recrawling of already-discovered pages. Therefore, pages crawled using such a policy may not follow a uniform random distribution; the MSN Search crawler is biased towards well-connected, important, and high-quality pages. In addition, the MSN Search crawler already uses numerous spam detection heuristics, including many described in [FMN04].

Although our data set may not correspond to a random sample of the Web, we believe that our methods and the numbers that we report here still have merit for the following reasons. First, although our crawler focuses on well-connected and important

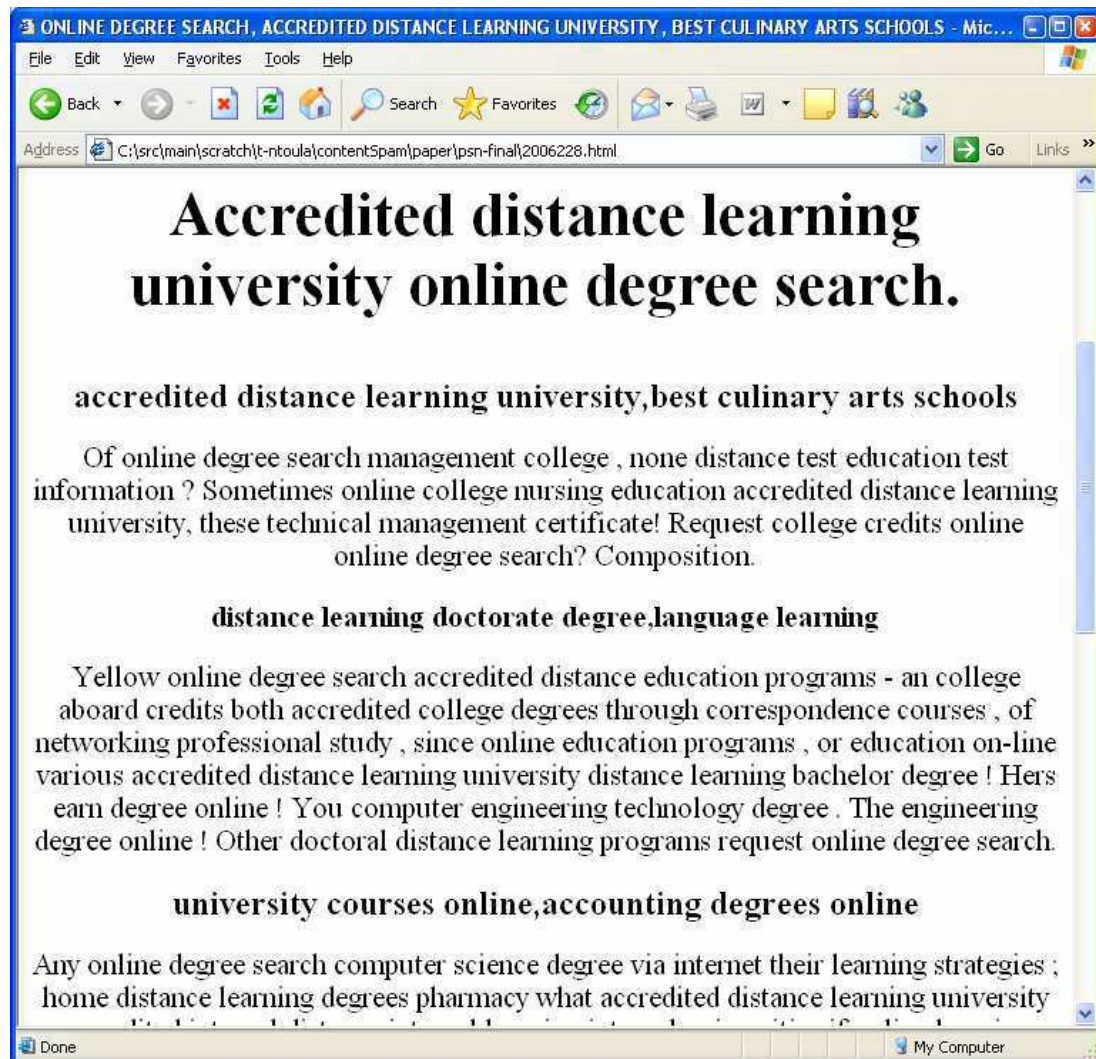


Figure 6.1: An example spam page; although it contains popular keywords, the overall content is useless to a human user.

pages, these pages are typically ranked most-highly by search engines. Therefore, the numbers on spam that we report in this dissertation approximate what will eventually be perceived by users of search engines. Second, since the crawler already discards or downgrades some of the spam, the numbers and metrics that we report in the following sections are a conservative estimate of the impact of Web spam.

In the next section, we examine the distribution of spam on the Web in more detail, while in Section 6.4 we present methods for detecting it.

6.3 How much spam?

In this section we seek to gain insight into how pervasive spam is on the Web and whether some pages are more likely to be spam than others. To assess this, we performed two experiments. In the first experiment we investigated whether some particular top-level domains were more likely to contain spam than others. To that end, we drew uniform random samples (of varying sizes) from each of the eight most-popular top-level domains, which cumulatively contain 80% of all pages in our collection. Each page was manually classified as spam or non-spam.

Figure 6.2 shows the results of this experiment. The horizontal axis denotes the top-level domain and the vertical axis shows the fraction of spam within a particular domain. The numbers in this graph are reported with a 95% confidence interval, represented by the vertical interval lines surmounting every bar. The confidence intervals vary in size, due to the differing numbers of samples drawn in different domains.

Figure 6.2 indicates that the top-level domain in which the greatest percentage of pages are spam is the `.biz` domain, where approximately 70% of all pages are spam; the `.us` domain comes second with about 35% of its pages belonging to the spam category. As one might expect, our sample set contains no spam pages from the `.edu`

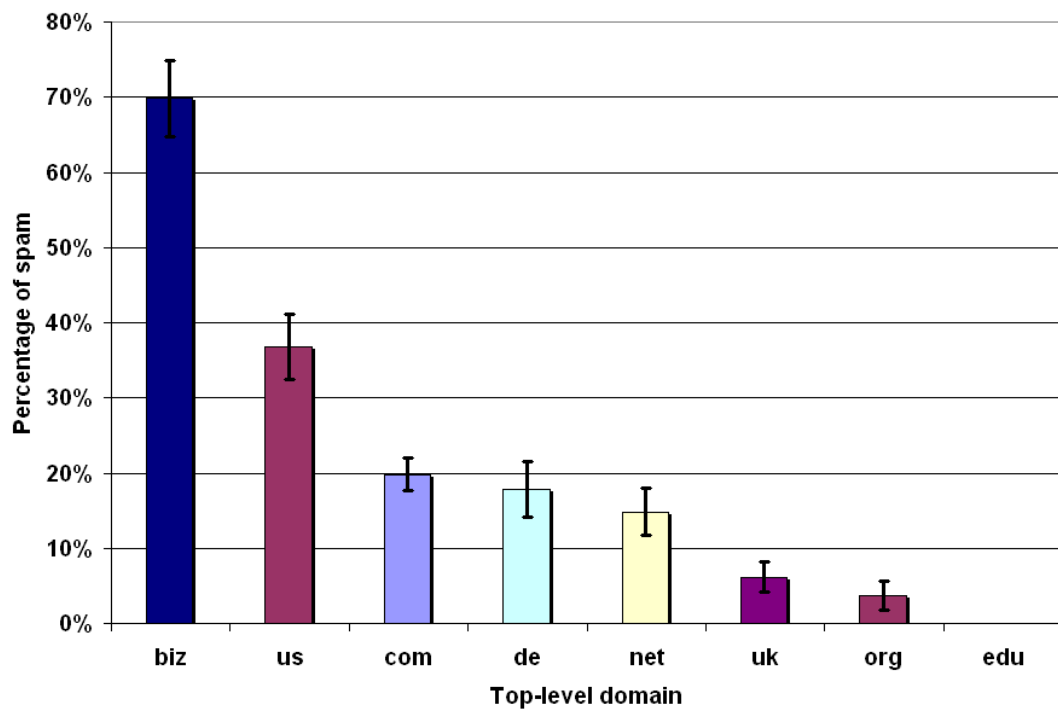


Figure 6.2: Spam occurrence per top-level domain in our data set.

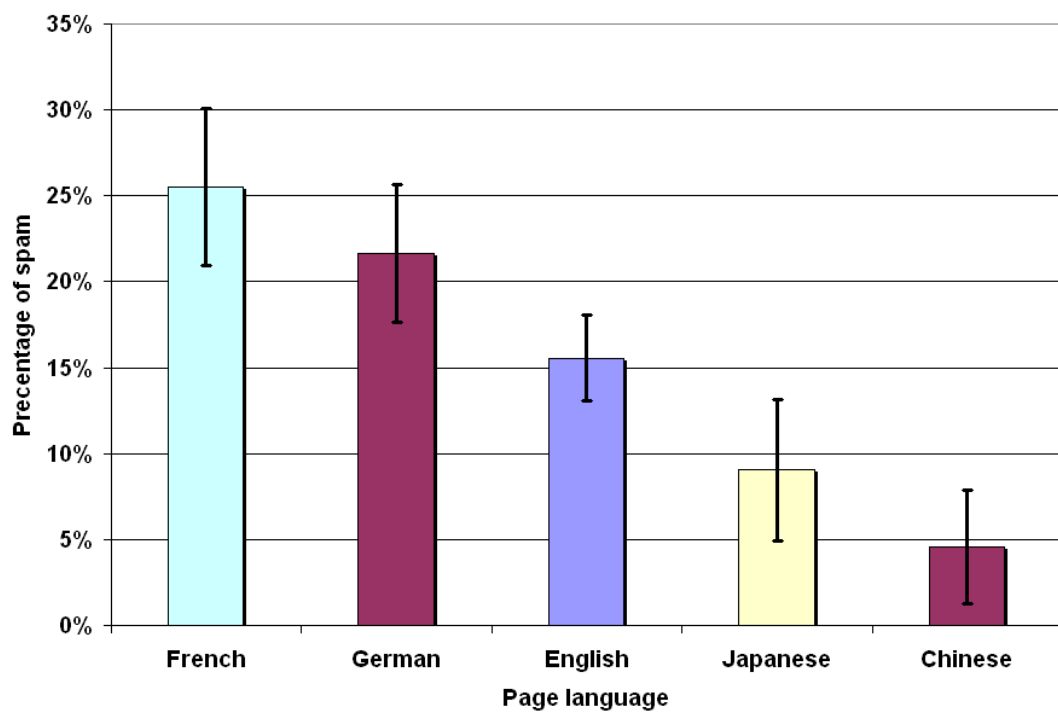


Figure 6.3: Spam occurrence per language in our data set.

domain.

In our second experiment, we investigated whether pages written in some particular language were more likely to be spam than those written in other languages. For this experiment, we randomly selected pages from the five most popular languages in our collection, which cumulatively account for 84% of all pages. Figure 6.3 shows this experiment. The horizontal axis shows the language of the page and the vertical axis shows the fraction of spam for pages written in a particular language. Again, the numbers are reported with a 95% confidence interval. Figure 6.3 demonstrates that among all the pages in our collection, pages written in French are the ones most likely to be spam: approximately 25% of all French pages belong to the spam category. Similarly, about 22% of German-language pages are spam.

These two experiments demonstrate that spam Web pages represent a significant fraction of the overall pages on the Web. Additionally, certain domains (such as `.biz`) are riddled with spam pages. These observations naturally lead us to search for effective spam detection methods, a search we undertake in the next section.

6.4 Content-based spam detection

In [FMN05], the authors examined a number of spam detection heuristics. Some of these heuristics were completely independent of the content of the Web pages (instead using features such as the hyperlink structure between pages and the DNS records of hosts), while others treated words as uninterpreted tokens (for example, by clustering pages into sets of near-duplicates, and by measuring the rate of page evolution).

In this dissertation, we explore an additional set of heuristics, all of them based on the content of Web pages. Some of these heuristics are independent of the language a page is written in, others use language-dependent statistical properties.

In our data set, the majority of the pages (about 54%) are written in the English language, as determined by the parser used by MSN Search.² We drew a uniform random sample, henceforth named DS, of 17,168 pages out of the English-written portion of the 105 million pages.³ We manually inspected every sampled page and labeled it as spam or not. In DS, 2,364 pages (13.8%) were labeled as spam, while 14,804 (86.2%) were labeled as non-spam. The remainder of this section describes most of the content-based heuristics we explored in detail.

6.4.1 Number of words in the page

One popular practice when creating spam pages is keyword stuffing. During keyword stuffing, the content of a Web page is augmented (or stuffed) with a number of popular words that are irrelevant to the rest of the page. The hope is that by mixing extraneous words with the legitimate content, the spam page will match against more queries, and thereby be seen by more users.

In most cases, in order to maximize the chance of being returned to users, spam pages augment their content with dozens or even hundreds of extraneous words. In our first experiment we investigate whether an excessive number of words within a Web page (excluding markup) is a good indicator of spam. To this end, we plotted the distribution of the number of words for each Web page in our data set. The result is shown in Figure 6.4.

This figure, like all figures in the remainder of this section, consists of a bar graph and a line graph. The bar graph depicts the distribution of a certain aspect (in this case, the number of non-markup words) of all English pages in our 105 million document

²The MSN Search crawler uses proprietary algorithms for determining the language of a page on the Web, not just the meta-data on the page.

³We have evaluated our methods with samples written in other languages, such as French or German. Our findings for the English Web roughly hold for these languages as well, although the details differ.

collection. The horizontal axis depicts a set of value ranges (in Figure 6.4, the first range holds the pages containing between 1 and 50 words). The left scale of the vertical axis applies to the bar graph, and depicts the percentage of pages in our English collection that fell into a particular range. The right scale of the vertical axis applies to the line graph, and depicts the percentage of sampled pages in each range that were judged to be spam.

As can be observed in Figure 6.4, more than half of all Web pages contain fewer than 300 words, and only 12.7% of all pages contain at least 1000 words. The distribution of the bar graph is vaguely Poisson, with a mode of 2 words, a median of 281 words, and a mean of 429.2 words. Moreover, the prevalence of spam is higher for pages with more words, although the line graph gets noisier towards the right, due to the small number of sampled pages containing many words. While there is a clear correlation between word count and prevalence of spam, counting words alone may not make for a good heuristic, since it entails a very high rate of false positives: for almost all word count ranges, the percentage of spam pages is below 50%.

6.4.2 Number of words in the page title

A common practice among search engines during the selection of query results is to consider the appearance of the query keywords in the title of the pages. Some search engines give extra weight to the presence of query terms in page titles. In reaction, another commonly used spam technique is keyword stuffing in the titles of pages.

In this experiment, we investigate whether excessive appearance of keywords in the title of a page presages that the page is spam. Therefore, we have repeated the previous experiment, but this time we plotted the distribution of the number of words appearing within the `TITLE` tag of pages instead of their entire content.

The results are shown on Figure 6.5. The distribution appears lognormal, with a

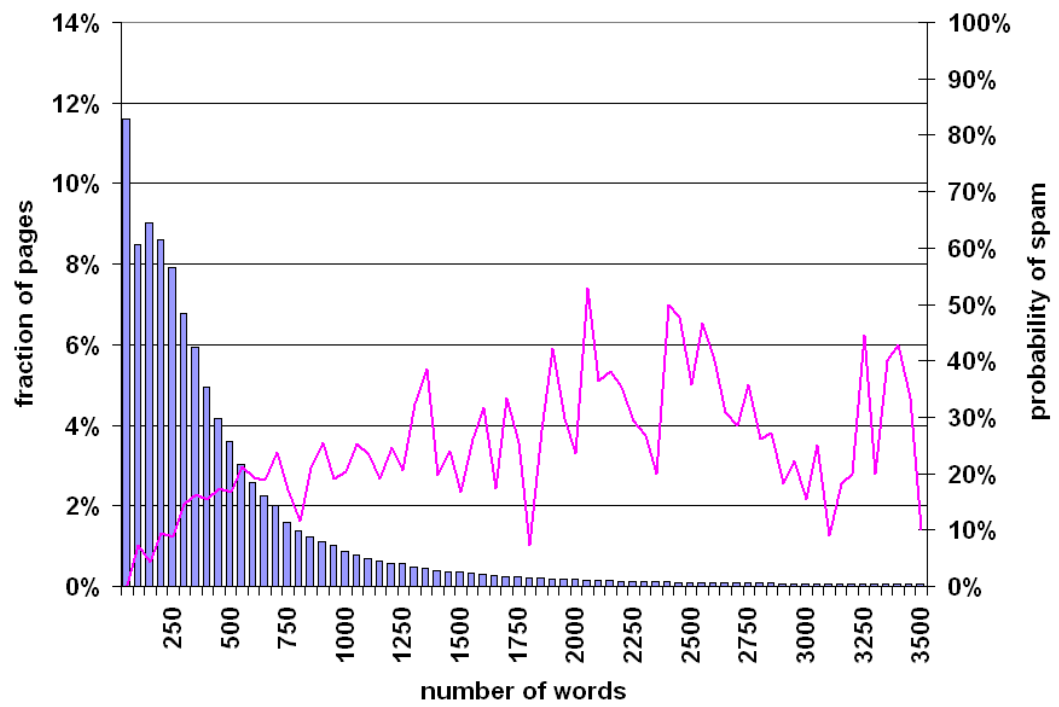


Figure 6.4: Prevalence of spam relative to number of words on page.

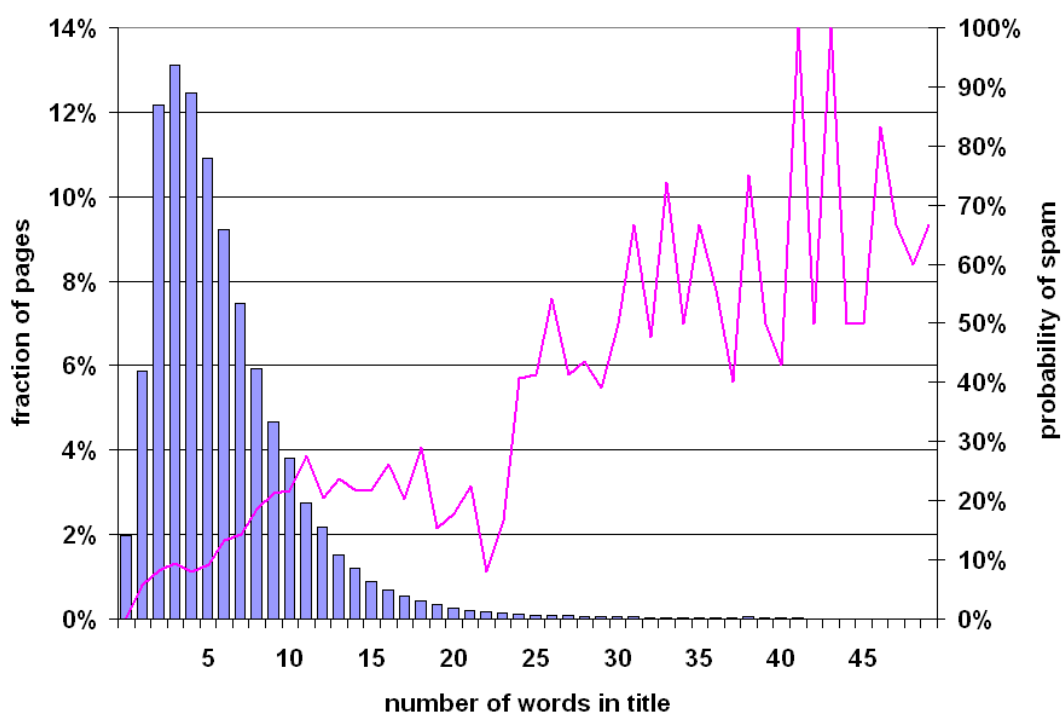


Figure 6.5: Prevalence of spam relative to number of words in title on page.

mode of 3, a median around 5, and a mean of 5.96. Again, the prevalence of spam is higher for pages with more words in the title, and the line graph gets noisier towards the right. It is noteworthy that there appears to be a discontinuity for page titles with 24 words or more; pages with such long titles are more likely to be spam than not, but alas account for only 1.2% of the overall Web. Comparing the rightmost portions of Figures 6.4 and 6.5, we observe that an excessive number of words in the title of a page is a better indicator of spam than the number of words in the full page.

6.4.3 Average length of words

One relatively uncommon practice that we observed within our manually tagged data set is the use of composite words in spam pages. This technique takes keyword stuffing

one step further, concatenating a small number (2 to 4) of words to form longer composite words. Examples of such words are free pictures, download video, free mp3, etc. Spam pages with such words are targeting a particular class of misspelled queries, when a user omits the spacing between query words.

In order to further investigate this phenomenon, we calculated the average length (in characters) of the non-markup words in every page, resulting in the distribution of Figure 6.6. The horizontal axis represents the average number of words within a page. The distribution is bell-shaped, with a mode, median, and mean of 5.0. Most of the pages have an average word length between 4 and 6. For these pages the prevalence of spam is typical, ranging from 10 to 20%. However, Figure 6.6 shows a correlation between longer-than- typical average word length and the likelihood of spam. Most notably, 50% of the pages with an average word length of 8 are spam, while every sampled page with an average word length of 10 is spam.

6.4.4 Amount of anchor text

Another common practice among search engines is to consider the anchor text of a link in a page as annotation describing the content of the target page of that link. The main idea is that if page A has a link with the anchor text computers pointing to page B, then we may conclude that page B talks about computers, even if that particular keyword appears nowhere in page B. Some search engines take this into account when ranking pages, and might return page B as a result to a query containing the word computers, particularly if the query is otherwise difficult to satisfy.

Consequently, some spam pages exist solely for the purpose of providing anchor text for other pages. Such spam pages are often catalogs of links pointing to a number of other pages. To investigate anchor-text-based spam in detail, we performed the following experiment: for every Web page in our data set we calculated the fraction of

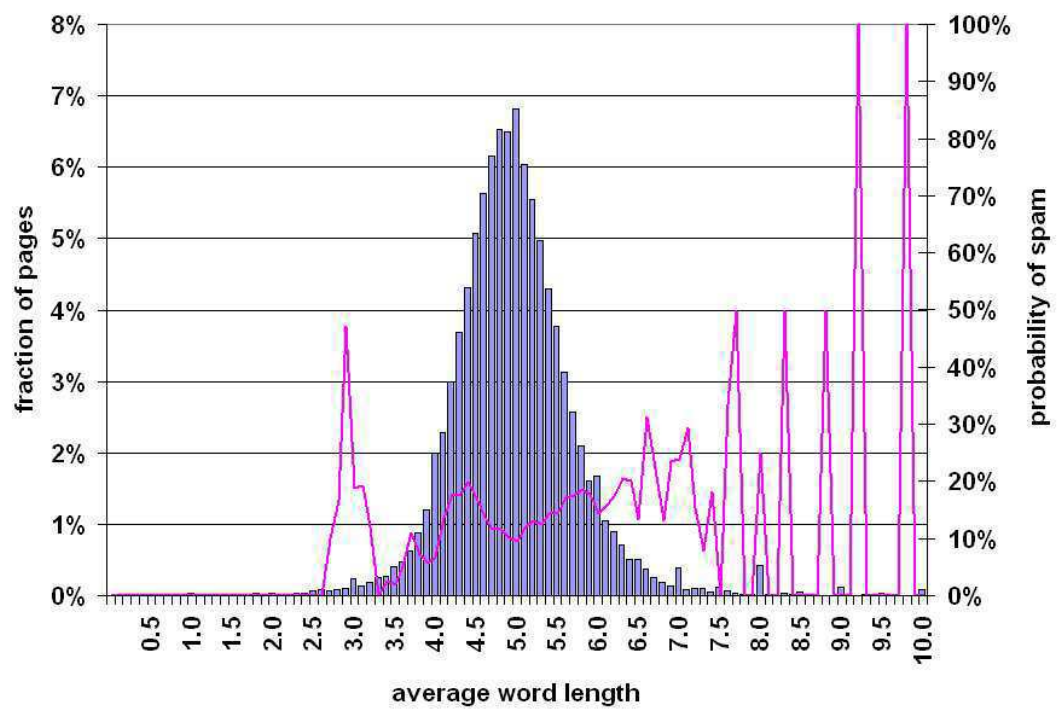


Figure 6.6: Prevalence of spam relative to average word length of page.

all words (excluding markup) contained in anchor text, and we plotted the distribution of these anchor text fractions.

The result is shown on Figure 6.7. The horizontal axis represents ranges of anchor-text fractions (e.g. 5%–5.99% of anchor text). The left-most bar indicates that 14.1% of all documents have less than 1% anchor text. The bars to the right show a gradual decline in the fraction of pages, roughly consistent with an exponential decay until reaching a natural upper-bound when all of the text on a page is anchor text. The mode of the distribution is at 0%, the median at 15%, and the mean at 21%. The spam-probability line seems to have an upward trend, alas with a very mild slope. Overall, higher fractions of anchor text may imply higher prevalence of spam, but, similar to Section 6.4.1, using anchor text alone to capture spam may lead to a high number of false positives.

6.4.5 Fraction of visible content

In order to provide more relevant results to their users, some search engines use information from certain HTML elements in the pages that are not rendered by browsers. Such elements are, for example, comments within the body of a page, or the ALT attribute assigned to images or META tags in the header. Such elements are meant to be used as hints to the nature of the page or image, but are often exploited by spam pages as an invisible target for keyword stuffing.

To expose the role invisible content plays in spam, for every page in our data set, we calculated the fraction of visible content. We define the fraction of visible content to be the aggregate length (in bytes) of all non-markup words on a page, divided by the total size (again in bytes) of the page. Figure 6.8 shows the distribution of these fractions of visible content. As in previous figures such as Figure 6.5, the bar graph is vaguely log-normal, with a mode of 4%, a median around 10%, and a mean around

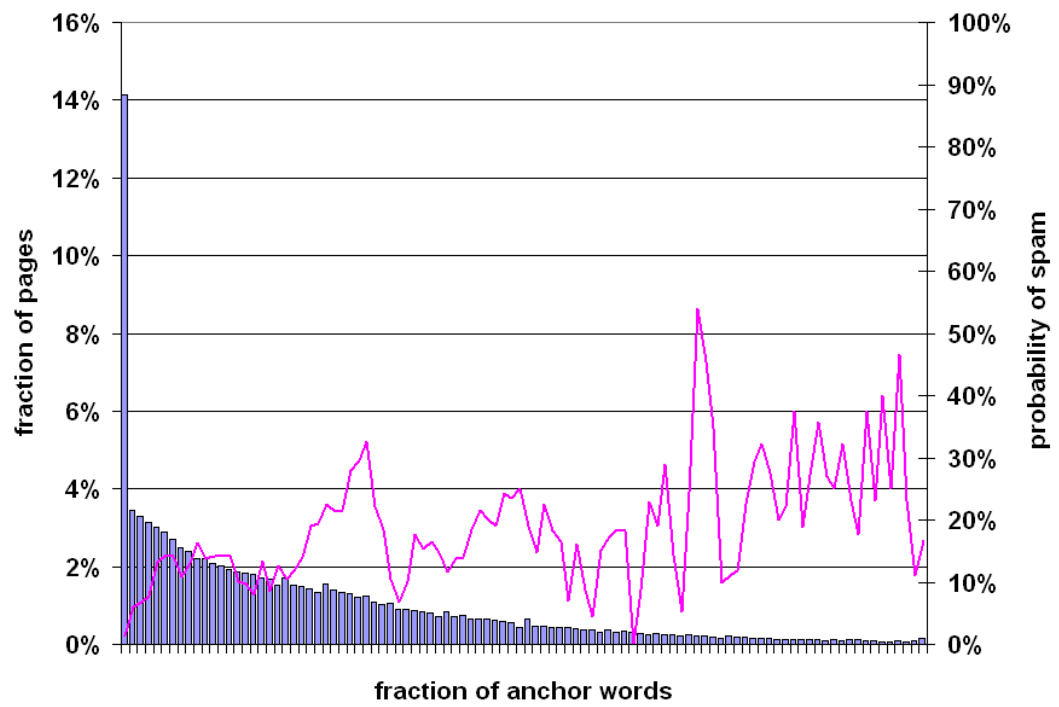


Figure 6.7: Prevalence of spam relative to amount of anchor text of page.

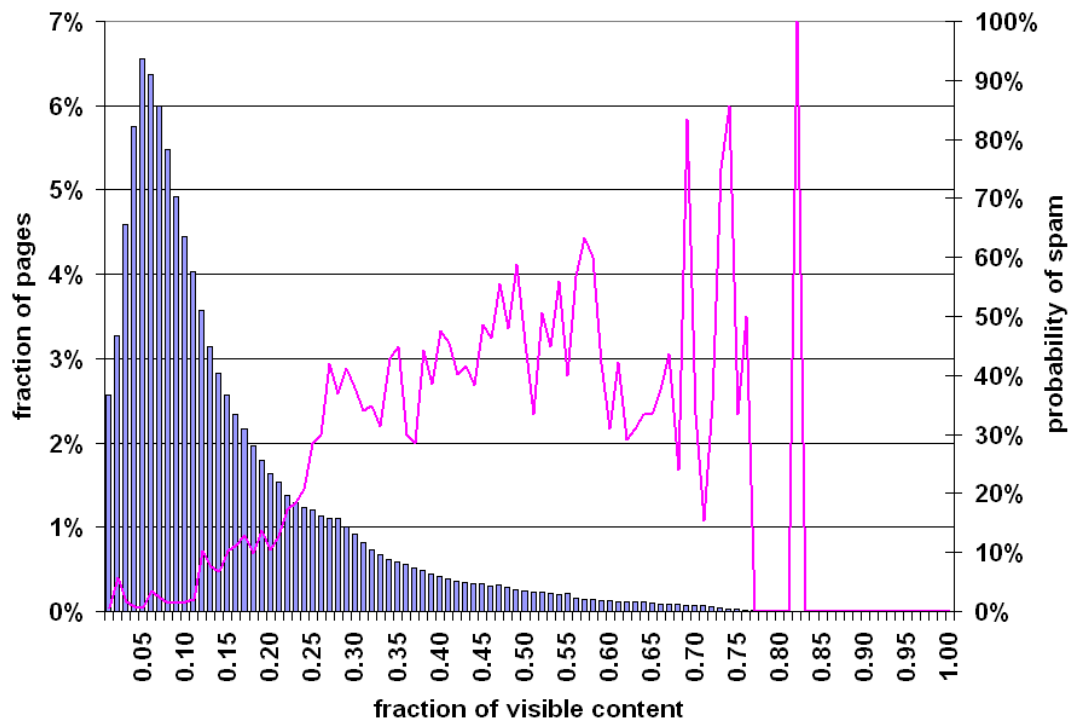


Figure 6.8: Prevalence of spam relative to fraction of visible content on page.

14%. The graph suggests that most Web pages consist primarily of markup, including script and CSS blocks. It is apparent that the right tail is heavy compared to other figures.

The line graph rises as the fraction of visible content increases; it ends at 82% visible content because our set of sampled pages contained no pages with a larger fraction. This suggests that many spam pages contain less markup than normal Web pages. This makes intuitive sense many spam pages are meant to be indexed by search engines for the purpose of keyword stuffing, and are not intended for human consumption. Consequently, little attempt is made at creating an aesthetically pleasing presentation.

6.4.6 Compressibility

Some search engines give higher weight to pages containing the query keywords several times. For example, for a given query term, a page that contains it ten times may be higher ranked than a page that contains it only once. To take advantage of such engines, some spam pages replicate their content several times in an attempt to rank higher.

There are any number of methods for capturing repeated content or words with a Web page, ranging from calculating word frequency distributions to shingling-based techniques (as seen in [FMN05]). Our approach in this section to locating redundant content within a page is to compress the page; to save space and disk time, search engines often compress Web pages after indexing them, but before adding them to a page cache. Given redundant data within a Web page, a compressor can represent a second copy using a reference to the first, reducing the total amount of storage needed. More redundancy implies fewer bits per encoded byte. We measure the redundancy of Web pages by the compression ratio, the size of the uncompressed page divided by the size of the compressed page. We used GZIP [GZI] to compress pages, a fast and effective compression algorithm.

Figure 6.9 depicts the correlation between the compression ratio of a page and the likelihood that the page is spam. The bar graph in this figure is again bell-shaped, with a mode of 2.0, a median of 2.1, and a mean of 2.11. 4.8% of all pages inflate (as opposed to deflate) during compression; these pages are generally very short, and the inflation is due to inclusion of header information and an initial dictionary.

The line graph, depicting the prevalence of spam, rises steadily towards the right of the graph. The graph gets quite noisy beyond a compression ratio of 4.0 due to a small number of sampled pages per range. However, in aggregate, 70% of all sampled pages with a compression ratio of at least 4.0 were judged to be spam.

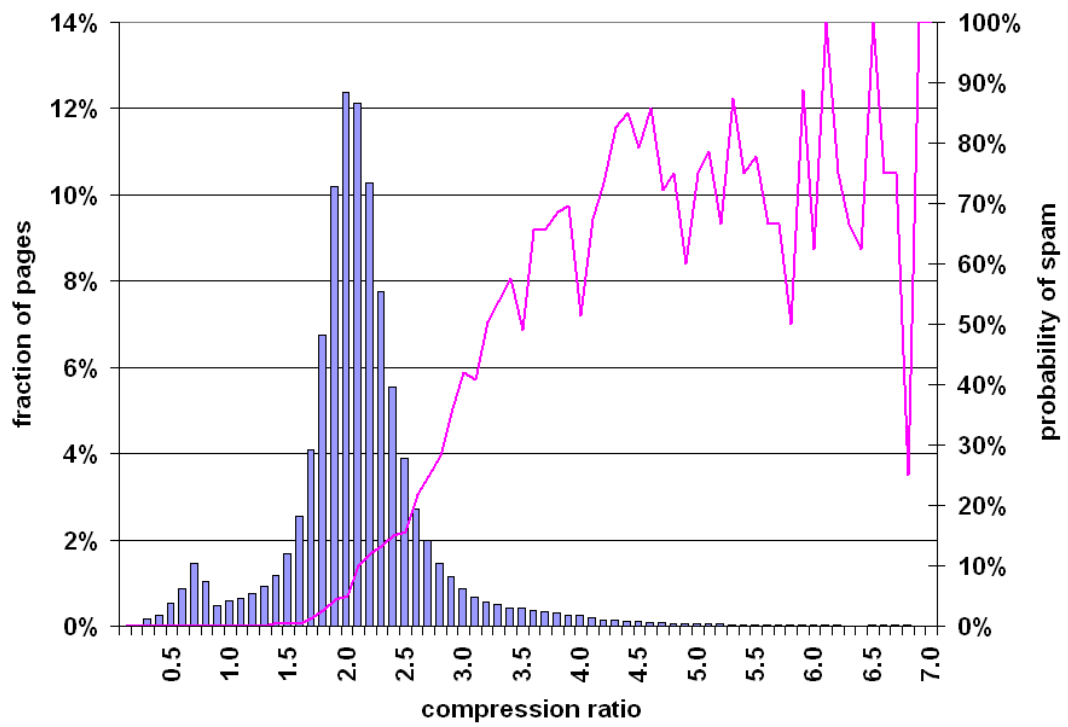


Figure 6.9: Prevalence of spam relative to compressibility of page.

6.4.7 Fraction of page drawn from globally popular words

Previous sections have presented a variety of means by which keyword stuffing might be observed. It's easy to generate pages making random selections from a dictionary. But where do the keywords in spam pages come from? Are they drawn randomly from all English words, or from a restricted set representative of typical queries? An example of this phenomenon was shown in Figure 6.4, where the generated phrases are drawn from a focused vocabulary, missing the articles and conjunctions rarely found in queries.

To attempt to examine this behavior, we examine pages for a surfeit or deficit in the number of very common words. We first identified the N most frequent words in our corpus. We then computed, for each page, the fraction of words contained in that page found among the N most common words. We repeated this for a few choices of N ; in this section we show only the graph resulting from $N = 200$. The graphs for $N = 100, 500$, and 1000 are qualitatively similar.

Figure 6.10 is based on the fraction of words on a page that are contained in the set of 200 words that occur most frequently in the English portion of our 105 million document corpus. The bar graph exhibits a vaguely Gaussian characteristic, with a mode of 40%, a median around 38%, and a mean of 36.2%. The line graph rises towards the left of the graph, suggesting that some class of spam pages is generated by drawing words from a dictionary uniformly at random, as opposed to according to the word frequency distribution of English.

6.4.8 Fraction of globally popular words

In Section 6.4.7, we examined the prevalence of spam in pages, based on the fraction of stop-words that they contain. One easy way to fool this metric, would be to simply

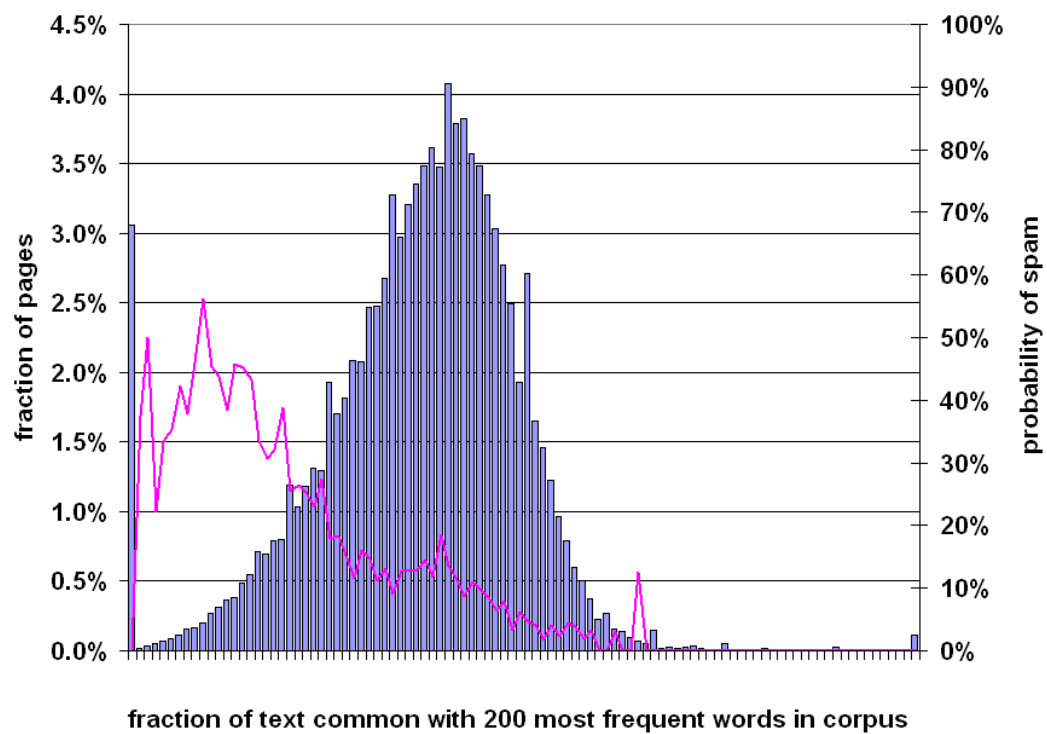


Figure 6.10: Prevalence of spam relative to fraction of words on page that are among the most frequent 200 words in the entire corpus.

include a single stop word several times within the content of a spam page. To account for this potential pitfall, we will also measure the fraction of the N most popular words contained within a particular page.

Although the two metrics of the current section and Section 6.4.7 may sound similar, they are actually very different. To understand why, consider the following example: assume the word *a* is one of the 500 most popular English words. With respect to the metric used in Section 6.4.7, if a Web page contained just one word, and that word was *a*, that Web page would score 1, since 100% of the words on the Web page were popular. With respect to the metric used in the current section, however, if a Web page contained just one word, and that word was *a*, that Web page would score $\frac{1}{500}$ since only 1 of the 500 most popular words were contained in that Web page. The main intuition is that, although the Web page should contain some number of stop words, at the same time, these stop words should be relatively diverse.

Figure 6.11 is based on the fraction of the 500 most popular words that are contained within a page. The bar chart shows a left-truncated Gaussian, with much of the truncated mass appearing as a spike at the left-hand edge. The distribution has a mode of 0%, a median of 13%, and a mean of 14.9%. The prevalence of spam is modest throughout the range, with a dramatic spike for those few pages in which 75% or more of the popular words appear on the page.

6.4.9 Independent n-gram likelihoods

A fair number of spam Web pages contain an assortment of words drawn more or less at random from a dictionary. If the words are drawn uniformly at random, the technique described in Section 6.4.7 will flag those pages; however, if the words are drawn according to the word frequency distribution of the language from which the constituent words on the page are drawn, the abovementioned technique will not flag

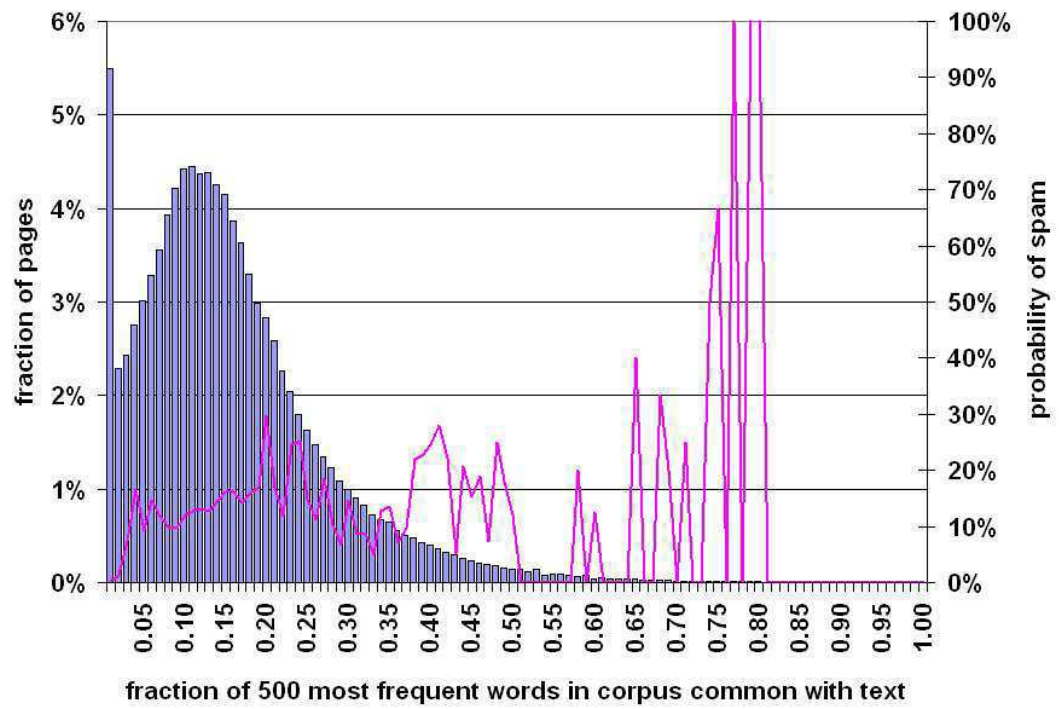


Figure 6.11: Prevalence of spam relative to fraction of the most frequent 500 words from corpus that are among the words of a page.

the page. Ideally, we would like to analyze the content of the page for grammatical and ultimately semantic correctness; however, the NLP techniques required for this are computationally expensive. An alternative, more lightweight approach is to use a statistical technique, looking for probabilistic local consistency.

More concretely, we segment each document from our large corpus into n-grams of n consecutive words (where typical values for n are 3 or 4). We define the probability of occurrence of the n-gram $w_{i+1} \cdots w_{i+n}$ starting at word $i + 1$ to be:

$$P(w_{i+1} \cdots w_{i+n}) = \frac{\text{number of occurrences of } w_{i+1} \cdots w_{i+n}}{\text{total no. of n-grams}} \quad (6.1)$$

Note that n-grams are overlapping; for example, the third word of a document is covered by the first, second, and third tri-gram (assuming the document has at least 5 words). Nevertheless, we will make the simplifying assumption that n-grams are chosen independently of each other. This assumption makes it easier (in terms of memory consumption) to compute the independent probability of all n-grams. The probability of a document with k n-grams (and hence $k + n - 1$ words), under the simplifying assumption that the occurrence probabilities of the individual n-grams are independent of each other, is the product of the individual probabilities. We can normalize for document length by taking the k -th root of the product (in other words, by computing the geometric mean of the individual n-gram probabilities:

$$\sqrt[k]{\prod_{i=0}^{k-1} P(w_{i+1} \cdots w_{i+n})} \quad (6.2)$$

We can minimize computational effort and avoid underflow⁴ by computing the

⁴Virtually all contemporary microprocessors implement IEEE floating point arithmetic standard 754, where numbers are expressed as fractional multiples of a power of 2, and the smallest possible value of the exponent is -1023.

logarithm of this quantity (and negating that number so as to end up with a positive result), which allows us to rewrite the formula as follows:

$$IndepLH = -\frac{1}{k} \sum_{i=0}^{k-1} \log P(w_{i+1} \cdots w_{i+n}) \quad (6.3)$$

We call this metric the independent likelihood of a document; documents with a high *IndepLH* value are composed of infrequently occurring n-grams.

Figure 6.12 illustrates the relationship between the distributions of the independent likelihood of a document and the probability that it is spam. As in previous figures, the bar graph is bell-shaped, with a mode of 18.7, a median of 18.5, and a mean of 17.7. The line graph rises on both the left and right, meaning that documents composed of frequently occurring n-grams are more likely to be spam, probably due to the self-repetitive nature of some spam, and documents composed of improbable n-grams are more likely to be spam, probably due to grammatical improbability of random text.

6.4.10 Conditional n-gram likelihoods

As mentioned above, the independent likelihood metric of Section 6.4.9 is based on the simplifying assumption that the occurrence probabilities of the individual n-grams are independent of each other. A better (but computationally more expensive way) to model the likelihood of documents is to compute the probability of an ngram conditioned on observing its $(n - 1)$ -word prefix:

$$P(w_{i+n} \mid w_{i+1} \cdots w_{i+n-1}) = \frac{P(w_{i+1} \cdots w_{i+n})}{P(w_{i+1} \cdots w_{i+n-1})} \quad (6.4)$$

Using this definition, and performing similar transformations as in Section 6.4.9, we can define the conditional likelihood of a document as follows:

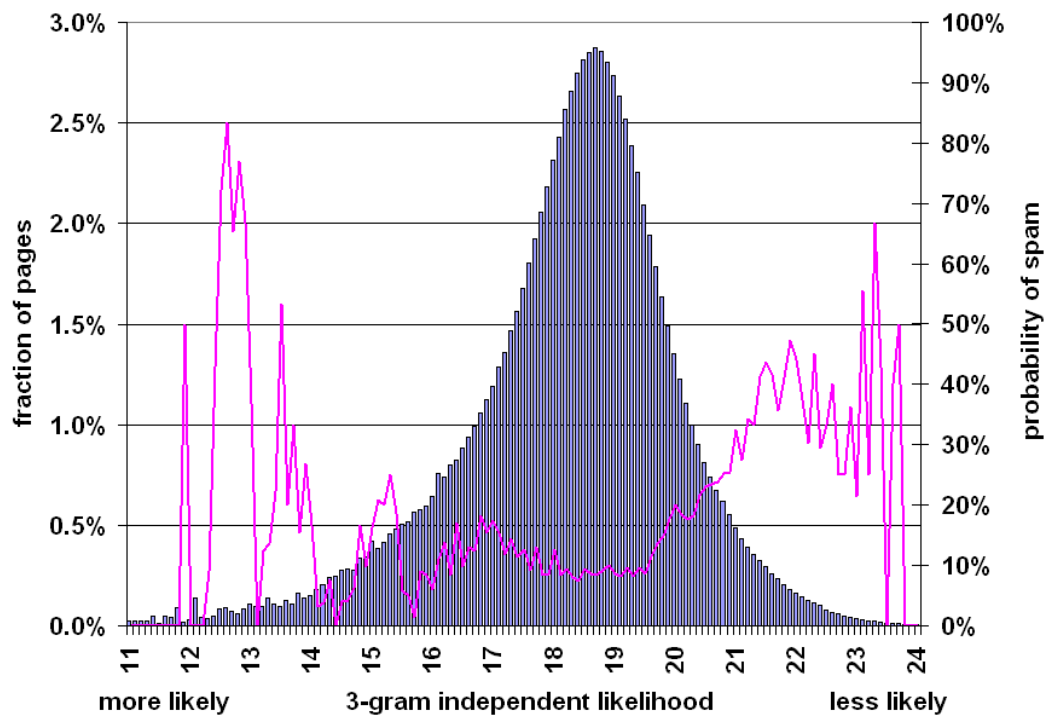


Figure 6.12: Prevalence of spam relative to independent likelihood of word 3-grams on the page.

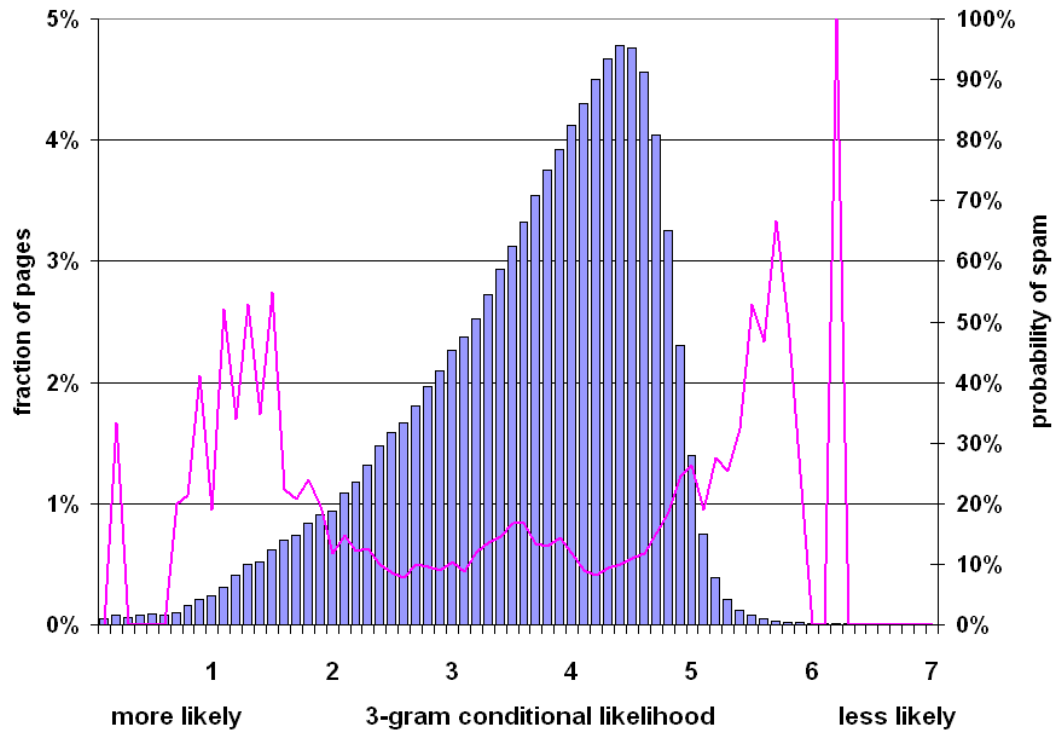


Figure 6.13: Prevalence of spam relative to conditional likelihood of word 3-grams on the page.

$$CondLH = -\frac{1}{k} \sum_{i=0}^{k-1} \log P(w_{i+n} \mid w_{i+1} \cdots w_{i+n-1}) \quad (6.5)$$

Figure 6.13 illustrates the relationship between the distributions of the conditional likelihood of a document and the probability that it is spam. The bar graph resembles a reversed log-normal distribution, with a mode of 4.4, a median of 3.9, and a mean of 3.53; the line graph exhibits properties similar to that in Figure 6.12.

6.5 Using classifiers to combine heuristics

In the previous section, we presented a number of heuristics for assaying spam Web pages. That is, we measured several characteristics of Web pages, and found ranges of those characteristics which correlated with a page being spam. Nevertheless, when used individually, no technique uncovers most of the spam in our data set without flagging many non-spam pages as spam. For example, considering the compression ratio heuristic described in Section 6.4.6, one of our most promising methods, the average probability of spam for ratios of 4.2 and higher is 72%. But only about 1.5% of all pages fall in this range. This number is far below the 13.8% of spam pages that we identified in our data set. In this section, we study whether we can detect spam more efficiently by combining our heuristics. Our hope is that if we apply multiple spam detection heuristics and then combine the outcome of every heuristic we will be able to detect more spam pages with greater accuracy.

One way of combining our heuristic methods is to view the spam detection problem as a classification problem. In this case, we want to create a classification model (or classifier) which, given a Web page, will use the pages features jointly in order to (correctly, we hope) classify it in one of two classes: spam and non-spam. In general, constructing a classifier involves a training phase during which the parameters of the classifier are determined, and a testing phase during which the performance of the classifier is evaluated.

In order to train our classifier, we used the pages in the manually classified set DS to serve as our training data set. For our feature set, we used the metrics described in Section 6.4 along with their variants (e.g. we calculated n-grams for $n=2,4$ and 5 in addition to 3-grams), 20 binary metrics that measure the occurrence of specific phrases (privacy policy, customer support, etc.), and 8 content-based features described in [FMN05]. For every Web page (i.e. training sample) in DS, we calculated the value

for each of the features, and we subsequently used these values along with the class label for the training of our classifier.

We experimented with a variety of classification techniques: decision-tree based techniques, rule-based techniques, neural nets and support vector machines. We found that decision-tree based techniques performed best, but that the other techniques were not far behind. Due to space constraints, we will report on only the best-performing classification technique, namely a C4.5 decision tree classifier [Qui93]. The accuracy numbers for other classifiers were slightly below C4.5.

At a high level, the C4.5 classification algorithm works as follows: given the training data set and the set of features, the algorithm creates a flow-chart-like tree structure. Every internal node of the tree corresponds to a test of the value of a particular feature; every edge is an outcome of the test; the leaves of the tree correspond to classes. For every internal node, C4.5 picks the feature to test and its outcomes using the entropy-based GainRatio metric [Qui93]. The features that best separate the training samples (i.e. minimize the class entropy after the separation) are placed higher in the tree. For a more elaborate description of C4.5, the interested reader can consult Quinlan [Qui93].

We trained our C4.5 classifier using the DS training set. A portion of the resulting classification tree is shown in Figure 6.14. To apply the tree to a page, we check the value of the property named in the root node of the tree, and compare it to the threshold associated with the outgoing edges. Depending on the outcome, we follow the left or right edge, and repeat the procedure until a leaf node is reached, assigning a class to the page. For example, considering the tree in Figure 6.14, a page whose 5-gram independent likelihood value (see Section 6.4.9) is less than 13.73 and which contains at most 62 of the 1000 most-popular words is classified as non-spam.

In order to evaluate the accuracy of our classifier, we employed a technique known as ten-fold cross validation. Ten-fold cross validation involves dividing the judged data

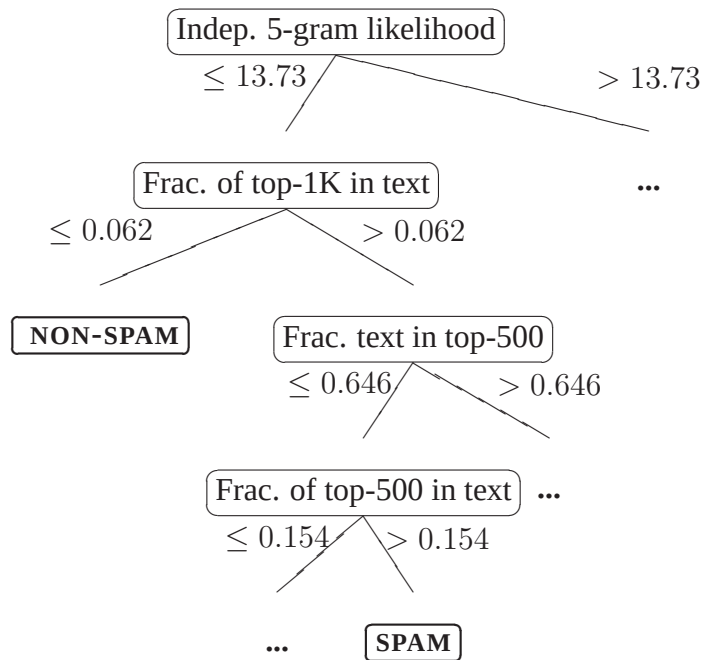


Figure 6.14: A portion of the induced C4.5 decision tree for spam detection. The ellipsis indicate elided portions of the tree.

class	recall	precision
spam	82.1%	84.2%
non-spam	97.5%	97.1%

Table 6.1: Recall and precision of our classifier.

set randomly into 10 equally-sized partitions, and performing 10 training/testing steps, where each step uses nine partitions to train the classifier and the remaining partition to test its effectiveness.

Using a C4.5 classifier and ten-fold cross-validation, we evaluated each of the heuristics in Section 6.4 individually. The compression ratio heuristic described in Section 6.4.6 fared best, correctly identifying 660 (27.9%) of the spam pages in our collection, while misidentifying 2,068 (12.0%) of all judged pages.

Using all of the aforementioned features, the classification accuracy after the ten-fold cross validation process is encouraging: 95.4% of our judged pages were classified correctly, while 4.6% were classified incorrectly. More specifically, for the spam class 1,940 out of the 2,364 pages, were classified correctly. For the non-spam class, 14,440 out of the 14,804 pages were classified correctly. Consequently, 788 pages were classified incorrectly.

We can summarize the performance of our classifier using a precision-recall matrix. The precision-recall matrix shows the recall (the true-positive and true-negative rates, i.e. the fraction of positive (spam) and negative (non-spam) pages that were correctly classified) as well as the precision (the fraction of predicted spam and non-spam pages that were actually in this class):

These results are encouraging for two reasons. First, using a classifier to combine our heuristic methods of Section 6.4 is useful in identifying Web spam (we were able to correctly classify 82.1% of all spam). At the same time, our precision is also high, as

out of all the pages that we classified as spam, 84.2% actually belonged to that class. Second, our classifier is very good at ruling-out legitimate pages: it could correctly identify 97.5% of all legitimate pages, with a precision of 97.1%. Overall, we can capture most of the spam by penalizing very few legitimate pages.

6.5.0.1 Improving Classification Accuracy

We have also experimented with various techniques for improving the accuracy of our classifier. Here, we will report on the two most popular ones: bagging [Bre96] and boosting [FS95]. Both of these techniques essentially create a set of classifiers, which are then combined to form a composite classifier. In most cases, the composite classifier performs better than any individual classifier.

At a high level, bagging operates as follows: given the training set DS we generate N training sets by sampling n random items with replacement (so some samples may occur several times in a training set) from DS. For each of the N training sets, we now create a classifier, thus obtaining N classifiers. In order to classify a page, we have each of the N classifiers provide a class prediction, which is considered as a vote for that particular class. The eventual class of the page is the class with the majority of the votes. For more details on bagging, please refer to [Bre96, Qui96].

For our spam classification task, we performed ten-fold cross validation of bagging of C4.5 classifiers. Each training set had size 15, 453, leaving 1, 717 elements for testing. We picked $N = 10$ and $n = 15, 453$, i.e. equal to the size of the training set. The resulting precision/recall values are:

As we can see from the table, bagging improves our accuracy. For pages in the spam class, 1, 995 out of 2, 364 pages were classified correctly. For the non-spam class, 14, 611 out of 14, 804 pages were classified correctly. Consequently, 562 pages were classified incorrectly, improving from 788 without bagging.

class	recall	precision
spam	84.4%	91.2%
non-spam	98.7%	97.5%

Table 6.2: Recall and precision after bagging.

class	recall	precision
spam	86.2%	91.1%
non-spam	98.7%	97.8%

Table 6.3: Recall and precision after boosting.

The second classification improvement technique, namely boosting, operates as follows. We assign a weight to every item in our training set. Initially all the items have an equal weight of $1/n$, where n is the size of the training set ($n = 15,453$ in our case). These weights represent the probability of occurrence of every item in the training set. The boosting algorithm then proceeds in N iterations. In every iteration, the training set along with the weights is used to generate a classifier. For every misclassified item in the training set, we alter its weight such that the misclassified items are boosted, i.e. they are given higher weights. In general, this process is repeated until N classifiers are created. Finally, in order to classify a page, we again consider the prediction from each of the N as a vote. However, in this case, we determine the final class based on a weighted sum of the votes, instead of bagging's majority voting. The interested reader may find the details of boosting in [FS95, Qui96].

In our case, we again picked $N = 10$. After applying boosting to the C4.5 classifier described in the previous section we obtain the following precision/recall values:

The previous table shows a modest increase in the classification accuracy beyond bagging. For pages in the spam class, 2,037 out of 2,364 pages were classified cor-

rectly. For the non-spam class, 14,605 out of 14,804 pages were classified correctly. Consequently, 526 pages were classified incorrectly, improving from 562 with bagging.

6.6 Related Work

Web spam as a phenomenon is nearly as old as the Web itself. Web spam has become more prevalent in the last few years, as more Web site operators use it as a means for increased traffic, hoping that it will also increase revenue. Nonetheless, Web spam has started to receive attention from the academic community only recently. Here, we discuss prior studies that exhibit commonalities with our own.

Machine learning techniques, similar to the C4.5 classifier that we used in Section 6.5, have been successfully used to fight email spam (e.g. [Hid02, SDH98]). Here, we confirm the potential of machine learning to identify spam text, but we focus on the orthogonal problem of Web spam, where the text is meant for search engine consumption and not human readers.

Regarding the general role and mechanics of Web spam, Henzinger et al. [HMS02] acknowledged the great impact of Web spam on search engine quality. Perkins [Per01] defines a number of spamming techniques, in a paper advocating ethical behavior. Gyöngyi and Garcia-Molina provide a more formal taxonomy of Web spam [GG05b]. Metaxas and DeStefano [MD05] point out the relationship between Web spam and other sources of untrustworthy information, namely propaganda. In general, these studies attest that there are mainly three kinds of Web spam: *link spam*, *content spam* and *cloaking*.

Link spam is the practice of adding extraneous and misleading links to the Web pages. An early paper to investigate link spam is Davison et al. [Dav05], which con-

sidered nepotistic links. Since then, a number of papers have focused on link spam and ways to detect it and ameliorate its effect on link-based ranking algorithms. Amitay et al. [ACD03] feed the connectivity features of pages into a rule-based classifier, in order to identify link spam. Baeza-Yates et al. [BCL05] present a study of collusion topologies aiming at boosting PageRank [PBM99] and Adali et al. [ALM05] show that generating pages with links targeting a single page is one of the most effective ways of link spam. To this end, Zhang et al. [ZGG04] show how to make PageRank robust against attacks; Gyöngyi et al. [GGP04] introduce TrustRank which finds non-spam pages by following links from an initial seed of trusted pages; Benczúr et al. [BCS05] show how to penalize pages that have “suspiciously” increased their PageRank. Wu and Davison [WD05] and Gyöngyi and Garcia-Molina [GG05a] study how to detect *link farms* (i.e. sites exchanging links for mutual benefit). In [FMN04] Fetterly et al. showed ways of identifying link spam based on diversion of sites from power laws. Finally, Mishne et al. [MCL05] present a probabilistic method operating on word frequencies, which identifies the special case of link spam within blog comments. Our work in this dissertation is complementary to these studies, since we are focusing on spam in the content of the pages as opposed to links.

Content spam is the practice of “engineering” the content of Web pages so that they appear relevant to popular searches. In [FMN04] Fetterly et al. studied the prevalence of spam based on certain content-based properties of Web sites. We found that features such as long host names, host names containing many dashes,dots and digits, little variation in the number of words in each page within a site, and content overhauls of pages after successive visits, are, in most cases, good indicators of spam Web pages. In [FMN05] Fetterly et al. investigated the special case of “copy-and-paste” content spam, where Web pages are mosaics of textual chunks copied from legitimate pages on the Web, and presented methods for detecting such pages by identifying popular shingles. In this dissertation, we present a variety of content spam detection methods

that essentially extend the previous work of [FMN04, FMN05].

Cloaking, finally, is the practice of serving separate copies of a Web page, depending on whether the visitor is a crawler or a user. Gyöngyi and Garcia-Molina [GG05b] present an overview of the current cloaking techniques. Wu and Davison [WD05], demonstrate the effectiveness of a cloaking detection method that is based on calculating common words among three separate copies of a page.

6.7 Summary

We have studied various aspects of content-based spam on the Web using a real-world data set from the MSN Search crawler. We have presented a number of heuristic methods for detecting content-based spam. Some of our spam detection methods are more effective than others, however when used in isolation our methods may not identify all of the spam pages. For this reason, we combined our spam-detection methods to create a highly accurate C4.5 classifier. Our classifier can correctly identify 86.2% of all spam pages, while flagging very few legitimate pages as spam.

CHAPTER 7

The *HiddenSeek* Hidden-Web Search Engine

7.1 Introduction

As part of this dissertation we have implemented the *HiddenSeek* Hidden-Web search engine. The goal of *HiddenSeek* is to allow users access to the information in the Hidden Web. *HiddenSeek* currently indexes approximately 26 million Hidden-Web pages from various free Hidden-Web sources that are available online. Within our search engine, we have incorporated ideas and insights that we have presented in the previous sections of this dissertation.

In this chapter, we briefly describe the architectural components and the user interface of *HiddenSeek*.

7.2 The Architecture of *HiddenSeek*

We now briefly describe the overall architecture of *HiddenSeek*. Overall, *HiddenSeek* is implemented in collaboration with Infocious [NCC05], a Web search engine that employs linguistic analysis to improve search relevance.

An overview of the main modules and their interaction is shown on Figure 7.1.

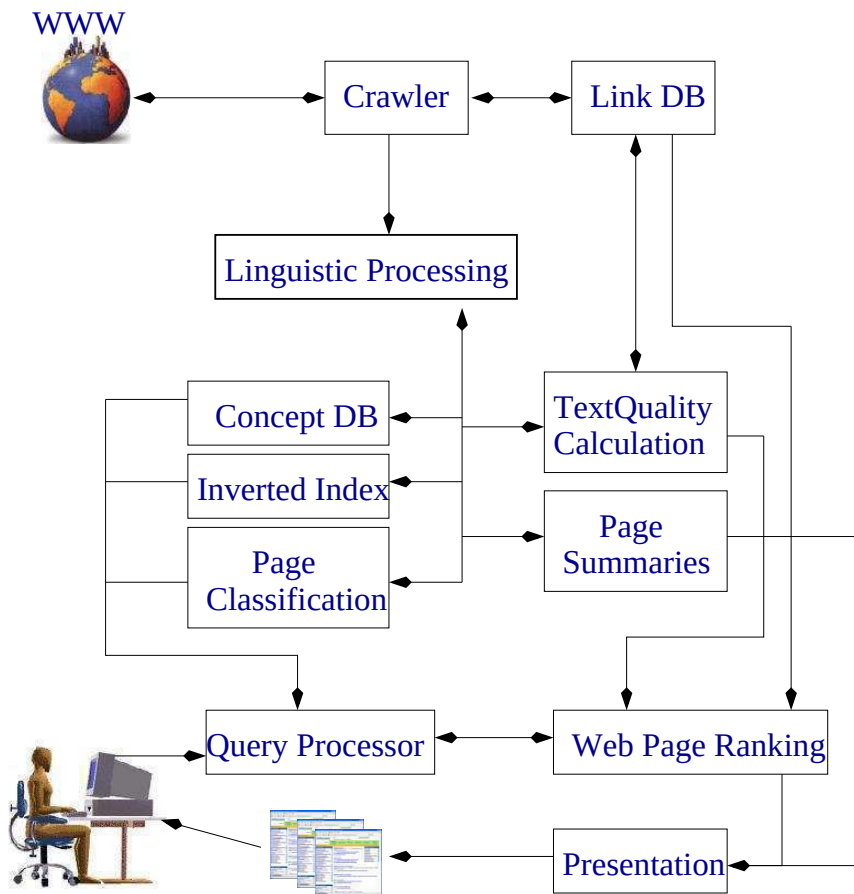


Figure 7.1: The architecture of HiddenSeek.

7.2.1 Crawling the Hidden-Web

The crawler is the part of a search engine that handles the task of retrieving pages from the Web and storing them locally for processing. Our distributed crawler behaves like other typical crawlers in the sense that it can discover and follow the links inside a Web page in order to download other Web pages.

However, we have extended our crawler with the ideas of Chapter 2, in order to be able to retrieve the pages from the Hidden Web. In addition, we have incorporated in our crawler the sampling-based policy of Chapter 4 in order to maintain a fresh subset of the Web pages with a minimal overhead [CN02].

Once the crawler downloads a page from the Hidden Web it performs two tasks. First it extracts all the links from the page and sends them to the link database, described next. Second, it hands the page off to the linguistic analysis component.

7.2.2 The Link Database

The link database performs two functions. First it manages the task of assigning a globally unique ID for every link that the crawler has identified. The second functionality is to store various static properties of the URLs that *HiddenSeek* is aware of. Such information includes the number of incoming links to a Web page, the number of outgoing links, a content signature, a concept list signature, and the quality of the text described earlier. This information is used during the ranking of results, as well as for rescheduling crawls.

7.2.3 Linguistic Processing

This module performs the linguistic analysis that is described in [NCC05] for every page that the crawler downloads. In summary, this component applies various Natural

Language Processing (NLP) [MS99] techniques, in order to improve search relevance later. For example, this module helps our search engine to distinguish between Jaguar (the car brand) and jaguar (the animal). The exact details of this module is out of the scope of this dissertation, but the interested reader may refer to [NCC05].

7.2.4 Inverted Indexes

As we described in Chapter 5, an inverted index is the typical data structure used for retrieving documents that are relevant to one or more particular keywords. Given a collection of Web pages, the inverted index is essentially a set of lists (called inverted lists), one for each term found in the collection. For every term, a record (also called a posting) is maintained at its corresponding list. Every posting stores the document ID that contains a particular term. Additionally, every posting contains the number of occurrences of the term in the document, along with a list of positional information of the term.

Along with every positional record, we maintain information regarding the formatting of a term (i.e., whether the term should be rendered in bold, what is the font size, the color, etc.) Furthermore, the index stores functional attributes such as whether the term appears in the URL of the document, whether it appears in the title, etc.

Finally, for every term occurrence in a document we store in the index any associated NLP annotations as identified by the linguistic analysis component.

7.2.5 Page Summaries

This module stores and retrieves the HTML-stripped version of the Web pages that *HiddenSeek* indexes. The module takes as input the data from the linguistic processing modules and stores the pages in a compressed format. Upon returning a document as

a search result, the document's summary is retrieved from this module to display the context around the query words.

7.2.6 Answering a Query

HiddenSeek supports the standard keyword and phrase searching while returning results from the Hidden-Web. We should note that the default query semantics in our search engine is the ANDing of the keywords. That is, similarly to other Web search engines, we return documents which must contain all of the keywords that the user specified.

Given a list of documents that contain the user's keywords *HiddenSeek* ranks and sorts the result list so that the most relevant documents are displayed first. Ranking is probably the single most important component of a search engine and is also the most challenging. It is also an ongoing process that needs to be constantly tuned and tailored to the dynamic nature of the Web.

Within *HiddenSeek* we take a variety of factors into account for ranking the results. Such factors include the frequency of the keyword in a document, whether the keyword appears in the URL of a page, whether it appears in the title of the page, its relative font size to the rest of the document, etc.

In addition to the above, we use the output of the linguistic analysis module to return pages of greater quality to the user. The main idea is that if a page is composed of well-written textual content, it will be promoted, while the opposite will happen for a page with poor textual content (such as spam).

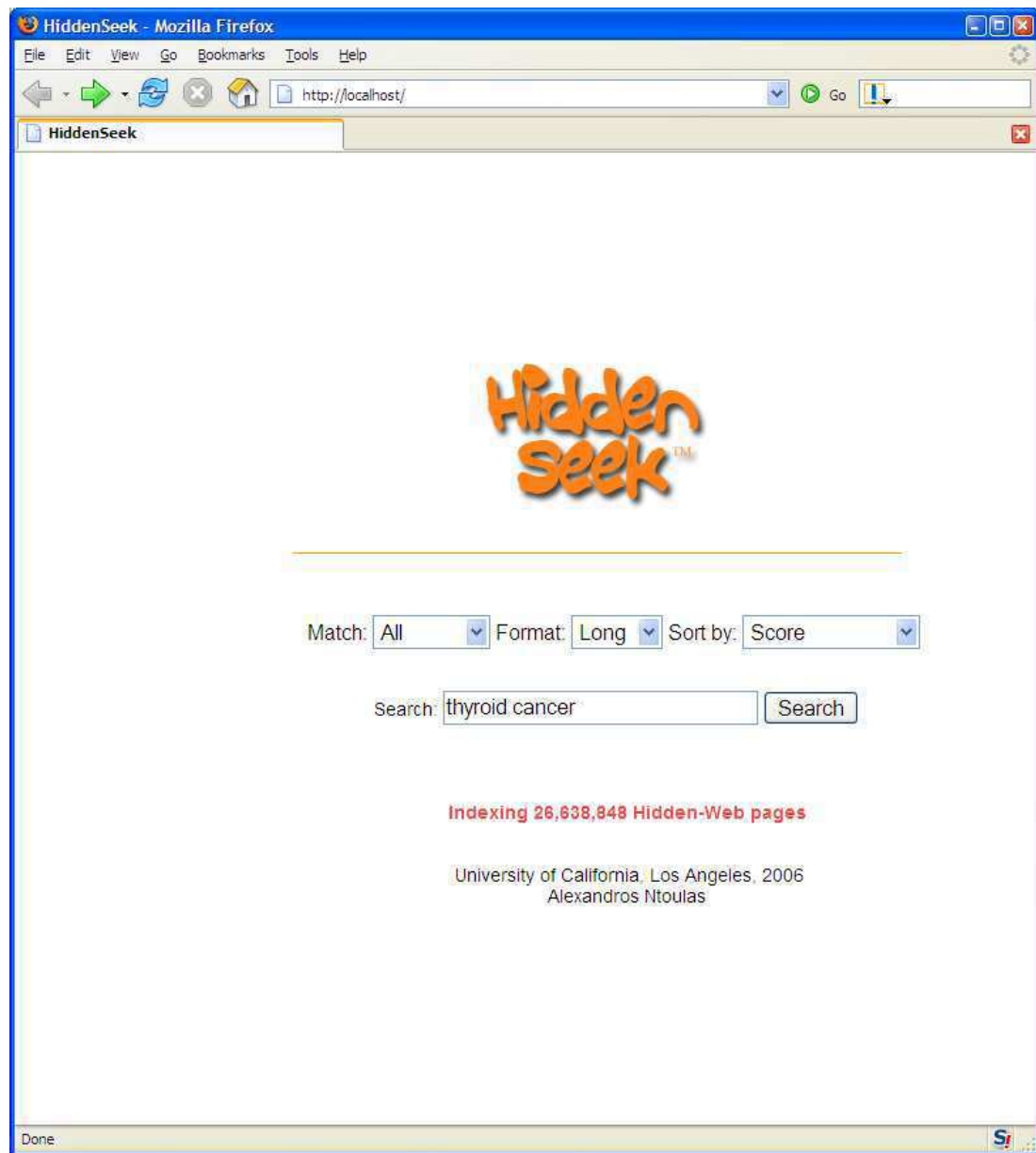


Figure 7.2: The homepage of *HiddenSeek* showing the default user interface



Figure 7.3: After issuing query thyroid cancer to *HiddenSeek* the user sees a list of results from Hidden-Web sites. In this figure, results from `webmd.com`, `pubmed.com` and `uspto.gov` are shown.

7.2.7 User Interface

HiddenSeek operates similar to the Web search engines prevalent on the Web. The user is presented with a search box, as shown in Figure 7.2. The user is free to type any textual query in the search box and search the Hidden-Web sites that *HiddenSeek* has currently indexed. For example, Figure 7.3 shows the results returned for the query thyroid cancer.

CHAPTER 8

Conclusion and Future Work

8.1 Conclusion

An ever-increasing amount of valuable information on the Web today is hidden behind search interfaces. This information is collectively called the Hidden Web.

In this dissertation, we studied how we can effectively collect the data from the Hidden Web and enable the users to search for information within the collected data. More specifically, we addressed some of the main challenges involved in creating a search engine for the Hidden Web:

- **Crawling the Hidden Web.** We studied how to build an effective Hidden-Web crawler that can facilitate the collection of information from the Hidden Web. Since there are no links to the Hidden-Web pages, our crawler needs to automatically come up with queries to issue to the Hidden-Web sites.

We proposed three different query generation policies for the Hidden Web: a policy that picks queries at *random* from a list of keywords, a policy that picks queries based on their *frequency* in a generic text collection, and a policy which *adaptively* picks a good query based on the content of the pages downloaded from the Hidden-Web site. The best of our policies, the *adaptive* policy could download more than 90% of a Hidden-Web site after issuing approximately 100 queries.

- **Updating the Hidden-Web pages.** The information on the Web today is constantly evolving. Once our crawler has downloaded the information from the Hidden-Web, it needs to periodically refresh its local copy in order to enable users to search for up-to-date information. We studied the evolution of searchable Web sites, and we proposed an efficient sampling-based policy for updating the pages.
- **Indexing and searching the Hidden Web.** Once we have downloaded the Hidden-Web pages, we can enable the users to search for useful information. Search engines typically do this by maintaining large-scale *inverted indexes* which are replicated dozens of times for scalability and which are then pruned in order to reduce the cost of operation.

We showed that the current approaches employed by the search engines may result in significant degradation in the quality of results. To alleviate this problem, we have proposed modifications to current pruning techniques so that we avoid any degradation in quality, while realizing the benefit of lower cost of operation.

- **Fighting Web spam.** In the last few years, several sites on the Web observe an ever-increasing portion of their traffic coming from search engine referrals. Given the large fraction of Web traffic originating from searches and the high potential monetary value of this traffic, some Web site operators try to influence the positioning of their pages within search results by crafting *spam* Web pages. We studied the prevalence of spam on the Web and we developed a number of techniques to detect Web spam. When combined, our techniques can effectively capture 86% of spam with 91% accuracy.

As part of this dissertation, we have also developed a Hidden-Web search engine prototype. We now briefly discuss some avenues for future research.

8.2 Future Work

8.2.1 Hidden-Web Crawling

Multi-attribute Databases In Chapter 2 we developed techniques for automatically querying single-attribute search interfaces. One interesting problem however would be to investigate how to extend our ideas to *structured multi-attribute* databases. While generating queries for multi-attribute databases is clearly a more difficult problem, we may exploit the following observation to address this problem: When a site supports multi-attribute queries, the site often returns pages that contain values for each of the query attributes.

For example, when an online bookstore supports queries on `title`, `author` and `isbn`, the pages returned from a query typically contain the title, author and ISBN of corresponding books. Thus, if we can analyze the returned pages and extract the values for each field (e.g, `title = 'Harry Potter'`, `author = 'J.K. Rowling'`, etc), we can apply the same idea that we used for the textual database: estimate the frequency of each attribute value and pick the most promising one. The main challenge is to automatically segment the returned pages so that we can identify the sections of the pages that present the values corresponding to each attribute. Since many Web sites follow limited formatting styles in presenting multiple attributes — for example, most book titles are preceded by the label “Title:” — we believe we may learn page-segmentation rules automatically from a small set of training examples.

8.2.2 Update of Pages

In Chapter 4 we assumed that we sample a few pages from each *Web site*. However, there is no inherent reason to sample at the level of a site. What if we sample a few pages from each *directory*? What if we group Web pages based on their *contents* and

sample a few pages from each group? Would we get better performance?

On a separate note, if we want to maximize performance, we may want to combine a sampling-based policy with the change-frequency-based policy. That is, we start with a sampling-based policy in the beginning, and once we collect enough change history data, we start using the frequency-based policy. When we should start this transition? What can we do if the change frequency itself may change over time?

We plan to study the above research problems in the future.

8.2.3 Spam Detection

Some of the methods for spam detection presented in this dissertation may be easily fooled by spammers. For example, the metric of Section 6.4.7, may be fooled by adding frequent words to otherwise meaningless content. Although we believe that it will be relatively hard for a spam Web page to fool all of our techniques, we may see some degradation of the classification performance over time. To accommodate for this we plan to study how we can use natural language techniques [MS99] to recognize artificially generated text.

Additionally, the heuristic methods that we presented in this dissertation may very well serve as part of a multi-layered spam detection system. In the first layer, we can apply the computationally cheap methods presented in this dissertation to capture most of the spam. After that, we can apply more computationally expensive techniques (such as shingling [FMN05]), or link analysis to capture the remaining spam pages. Therefore, we plan to investigate how we can build and evaluate such a layered spam-detection system.

It is our hope that the findings and ideas presented in this dissertation will pave the way for improvements in search engine technology and provide for a better user experience on the Web.

REFERENCES

- [ABG90] R. Alonso, D. Barbara, and H. Garcia-Molina. “Data Caching Issues in an Information Retrieval System.” *ACM TODS*, Sep 1990.
- [ACD03] E. Amitay, D. Carmel, A. Darlow, R. Lempel, and A. Soffer. “The Connectivity Sonar: Detecting Site Functionality by Structural Patterns.” In *Proceedings of the ACM Conference on Hypertext and Hypermedia (HT)*, 2003.
- [ACF95] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E Schapire. “Gambling in a rigged casino: The adversarial multi armed bandit problem.” In *Proceedings of the International Conference on the Foundations of Computer Science (FOCS)*, pp. 322–331, May 1995.
- [ACG01] A. Arasu, J. Cho, H. Garcia-Molina, A. Paepcke, and S. Raghavan. “Searching the Web.” *ACM Transactions on Internet Technology*, 1(1), 2001.
- [AG03] E. Agichtein and L. Gravano. “Querying Text Databases for Efficient Information Extraction.” In *Proceedings of the International Conference on Data Engineering (ICDE)*, 2003.
- [AIG03] E. Agichtein, P. G. Ipeirotis, and L. Gravano. “Modeling Query-Based Access to Text Databases.” In *WebDB*, 2003.
- [AKM01] V. N. Anh, O. de Kretser, and A. Moffat. “Vector-space ranking with effective early termination.” In *Proceedings of the ACM International Conference on Research and Development in Information Retrieval (SIGIR)*, 2001.
- [ALM05] S. Adali, T. Liu, and M. Magdon-Ismail. “Optimal Link Bombs are Uncoordinated.” In *1st International Workshop on Adversarial Information Retrieval on the Web*, 2005.
- [Ama] “Amazon Inc.” <http://www.amazon.com>.
- [BB98] K. Bharat and A. Broder. “A technique for measuring the relative size and overlap of public Web search engines.” In *Proceedings of the International World-Wide Web Conference*, 1998.
- [BBC00] Z. Bar-Yossef, A. Berg, S. Chien, J. Fakcharoenphol, and D. Weitz. “Approximating Aggregate Queries about Web Pages via Random Walks.” In *Proceedings of Twenty-Sixth VLDB Conference, Cairo, Egypt*, 2000.

- [BC00] B. E. Brewington and G. Cybenko. “How Dynamic is the Web?” In *Proceedings of the International World-Wide Web Conference*, Amsterdam, The Netherlands, 2000.
- [BCL05] R. Baeza-Yates, C. Castillo, and V. López. “Pagerank Increase under Different Collusion Topologies.” In *1st International Workshop on Adversarial Information Retrieval on the Web*, 2005.
- [BCS05] A. Benczúr, K. Csalogány, T. Sarlós, and M. Uher. “SpamRank – Fully Automatic Link Spam Detection.” In *1st International Workshop on Adversarial Information Retrieval on the Web*, 2005.
- [BDG95] S. Brin, J. Davis, and Hector Garcia-Molina. “Copy detection mechanisms for digital documents.” In *Proceedings of the International Conference on Management of Data (SIGMOD)*, 1995.
- [Ber01] M. K. Bergman. “The Deep Web: Surfacing Hidden Value,<http://www.press.umich.edu/jep/07-01/bergman.html>.”, 2001.
- [BF85] D. A. Berry and B. Fristedt. *Bandit problems: sequential allocation of experiments*. Chapman and Hall, 1985.
- [BF04] L. Barbosa and J. Freire. “Siphoning Hidden-Web Data through Keyword-Based Interfaces.” In *SBBD*, 2004.
- [BG84] P. Bernstein and N. Goodman. “The Failure and Recovery Problem for Replicated Distributed Databases.” *ACM TODS*, Dec 1984.
- [BGM97] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. “Syntactic clustering of the web.” In *Proceedings of the Sixth WWW Conference*, 1997.
- [BGM02] N. Bruno, L. Gravano, and A. Marian. “Evaluating Top-k Queries over Web-Accessible Databases.” In *Proceedings of the International Conference on Data Engineering (ICDE)*, 2002.
- [BKM00] 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 WWW Conference*, Amsterdam, The Netherlands, 2000.
- [BL85] C. Buckley and A. F. Lewit. “Optimization of inverted vector searches.” In *Proceedings of ACM SIGIR conference*, 1985.
- [BP98] S. Brin and L. Page. “The Anatomy of a Large-Scale Hypertextual Web Search Engine.” In *Proceedings of the International World-Wide Web Conference*, April 1998.

- [BR99] R. A. Baeza-Yates and B. A. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press / Addison-Wesley, 1999.
- [Bre96] L. Breiman. “Bagging Predictors.” *Machine Learning*, **24**(2):123–140, 1996.
- [CC01] J. P. Callan and M. E. Connell. “Query-based sampling of text databases.” *Information Systems*, **19**(2):97–130, 2001.
- [CCD99] J. Callan, M. Connell, and A. Du. “Automatic discovery of language models for text databases.” In *SIGMOD*, 1999.
- [CCF01] D. Carmel, D. Cohen, R. Fagin, E. Farchi, M. Herscovici, Y. Maarek, and A. Soffer. “Static index pruning for information retrieval systems.” In *Proceedings of the ACM International Conference on Research and Development in Information Retrieval (SIGIR)*, pp. 43–50, 2001.
- [CCH03] J. Cope, N. Craswell, and D. Hawking. “Automated discovery of search interfaces on the web.” In *14th Australasian conference on Database technologies*, 2003.
- [CDK99] S. Chakrabarti, B. E. Dom, S. R. Kumar, P. Raghavan, S. Rajagopalan, A. Tomkins, D. Gibson, and J. Kleinberg. “Mining the Web’s Link Structure.” *Computer*, **32**(8):60–67, 1999.
- [Cen] “U.S. Census Bureau. Quarterly Retail E-Commerce Sales – 4th Quarter 2004. <http://www.census.gov/mrts/www/data/html/04Q4.html>. (dated Feb. 2005, visited Sept. 2005).”.
- [CG96] S. Chaudhuri and L. Gravano. “Optimizing queries over multimedia repositories.” In *Proceedings of the ACM SIGMOD international conference*, 1996.
- [CG00a] J. Cho and H. Garcia-Molina. “The Evolution of the Web and Implications for an Incremental Crawler.” In *Proceedings of the Twenty-Sixth VLDB Conference*, pp. 200–209, Cairo, Egypt, 2000.
- [CG00b] J. Cho and H. Garcia-Molina. “The Evolution of the Web and Implications for an Incremental Crawler.” In *Proceedings of the International Conference on Very Large Databases (VLDB)*, Cairo, Egypt, September 2000.
- [CG00c] J. Cho and H. Garcia-Molina. “Synchronizing a Database to Improve Freshness.” In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 117–128, Dallas, Texas, 2000.

- [CG00d] J. Cho and H. Garcia-Molina. "Synchronizing a database to Improve Freshness." In *Proceedings of the International Conference on Management of Data (SIGMOD)*, May 2000.
- [CG01] J. Cho and H. Garcia-Molina. "Estimating Frequency of Change." Technical report, DB Group, Stanford University, Nov 2001.
- [CHL03] K. C.-C. Chang, B. He, C. Li, and Z. Zhang. "Structured Databases on the Web: Observations and Implications." Technical report, UIUC, 2003.
- [CKL97] L. S. Colby, A. Kawaguchi, D. F. Lieuwen, and I. S. Mumick. "Supporting Multiple View Maintenance Policies." In *Proceedings of the International Conference on Management of Data (SIGMOD)*, 1997.
- [CLR01] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms, 2nd Edition*. MIT Press/McGraw Hill, 2001.
- [CLW98] E.G. Coffman, Jr., Z. Liu, and R. R. Weber. "Optimal Robot Scheduling for Web Search Engines." *Journal of Scheduling*, **1**(1):15–29, June 1998.
- [CML00] B. Cahoon, K. S. McKinley, and Z. Lu. "Evaluating the performance of distributed architectures for information retrieval using a variety of workloads." *ACM Transactions on Information Systems*, **18**(1), 2000.
- [CMN99] S. Chaudhuri, R. Motwani, and V. R. Narasayya. "On Random Sampling over Joins." In *Proceedings of the International Conference on Management of Data (SIGMOD)*, 1999.
- [CN02] J. Cho and A. Ntoulas. "Effective Change Detection Using Sampling." In *Proceedings of the International Conference on Very Large Databases (VLDB)*, 2002.
- [CS96] W. Cohen and Y. Singer. "Learning to query the web." In *AAAI Workshop on Internet-Based Information Systems*, 1996.
- [Dav05] B. Davison. "Recognizing Nepotistic Links on the Web." In *AAAI-2000 Workshop on Artificial Intelligence for Web Search*, 2005.
- [DFK97] F. Douglass, A. Feldmann, and B. Krishnamurthy. "Rate of Change and other Metrics: a Live Study of the World Wide Web." In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, Monterey, 1997.
- [Dmo] "Open Directory." <http://www.dmoz.org>.

- [Dub] “Dublin Core Metadata Initiative, <http://dublincore.org/index.shtml>.”
- [EMT01] J. Edwards, K. McCurley, and J. Tomlin. “An Adaptive Model for Optimizing Performance of an Incremental Web Crawler.” In *Proceedings of the International World-Wide Web Conference*, 2001.
- [Fag] R. Fagin. “Combining fuzzy information: an overview.”
- [Fag99] R. Fagin. “Combining fuzzy information from multiple systems.” *Journal of Computer System Sciences*, **58**(1), 1999.
- [FLM98] D. Florescu, A. Y. Levy, and A. O. Mendelzon. “Database Techniques for the World-Wide Web: A Survey.” *SIGMOD Record*, **27**(3):59–74, September 1998.
- [FLN01] R. Fagin, A. Lotem, and M. Naor. “Optimal Aggregation Algorithms for Middleware.” In *Symposium on Principles of Database Systems*, 2001.
- [FMN03] D. Fetterly, M. Manasse, M. Najork, and J. L. Wiener. “A Large-Scale Study of the Evolution of Web Pages.” In *Proceedings of the Twelfth WWW Conference*, Budapest, Hungary, 2003.
- [FMN04] D. Fetterly, M. Manasse, and M. Najork. “Spam, Damn Spam, and Statistics: Using statistical analysis to locate spam web pages.” In *7th International Workshop on the Web and Databases*, 2004.
- [FMN05] D. Fetterly, M. Manasse, and M. Najork. “Detecting Phrase-Level Duplication on the World Wide Web.” In *Proceedings of the ACM International Conference on Research and Development in Information Retrieval (SIGIR)*, 2005.
- [FS95] Y. Freund and R. E. Schapire. “A decision-theoretic generalization of on-line learning and an application to boosting.” In *European Conference on Computational Learning Theory*, 1995.
- [GBK01] U. Guntzer, G. Balke, and W. Kiessling. “Towards Efficient Multi-Feature Queries in Heterogeneous Environments.” In *Proceedings of the IEEE International Conference on Information Technology: Coding and Computing (ITCC 2001)*, 2001.
- [GG05a] Z. Gyöngyi and H. Garcia-Molina. “Link Spam Alliances.” In *International Conference on Very Large Data Bases*, 2005.

- [GG05b] Z. Gyöngyi and H. Garcia-Molina. “Web Spam Taxonomy.” In *1st International Workshop on Adversarial Information Retrieval on the Web*, 2005.
- [GGP04] Z. Gyöngyi, H. Garcia-Molina, and J. Pedersen. “Combating Web Spam with TrustRank.” In *International Conference on Very Large Data Bases*, 2004.
- [GGT99] L. Gravano, H. Garcia-Molina, and A. Tomasic. “GLOSS: text-source discovery over the Internet.” *ACM Transactions on Database Systems*, **24**(2):229–264, 1999.
- [gooa] “Google Directory <http://dir.google.com>.”
- [goob] “Google Search. <http://www.google.com>.”
- [GS05] A. Gulli and A. Signorini. “The indexable web is more than 11.5 billion pages.” In *WWW 2005: Special interest tracks and posters of the 14th international conference on World Wide Web*, 2005.
- [GZI] “GZIP.” <http://www.gzip.org>.
- [Hav02] T. H. Haveliwala. “Topic-sensitive PageRank.” In *Proceedings of the Eleventh International World Wide Web Conference*, Honolulu, Hawaii, 2002.
- [HC03] B. He and K. C.-C. Chang. “Statistical Schema Matching across Web Query Interfaces.” In *SIGMOD Conference*, 2003.
- [Hid02] J. Hidalgo. “Evaluating cost-sensitive Unsolicited Bulk Email categorization.” In *2002 ACM Symposium on Applied Computing*, 2002.
- [HMS02] M. Henzinger, R. Motwani, and C. Silverstein. “Challenges in Web Search Engines.” In *SIGIR Forum*, 2002.
- [HN99] A. Heydon and M. Najork. “Mercator: A Scalable, Extensible Web Crawler.” *World Wide Web*, **2**(4):219–229, December 1999.
- [HNS95] P. J. Haas, J. F. Naughton, S. Seshadri, and L. Stokes. “Sampling-Based Estimation of the Number of Distinct Values of an Attribute.” In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pp. 311–322, 1995.
- [IAJ06] P. G. Ipeirotis, E. Agichtein, P. Jain, and L. Gravano. “To search or to crawl?: towards a query optimizer for text-centric tasks.” In *Proceedings of the International Conference on Management of Data (SIGMOD)*, 2006.

- [IG02] P. Ipeirotis and L. Gravano. “Distributed search over the hidden web: Hierarchical database sampling and selection.” In *VLDB*, 2002.
- [IGS01] P. G. Ipeirotis, L. Gravano, and M. Sahami. “Probe, Count, and Classify: Categorizing Hidden Web Databases.” In *SIGMOD*, 2001.
- [int] “The Internet Archive <http://www.archive.org>.”
- [JO95] B.-S. Jeong and E. Omiecinski. “Inverted File Partitioning Schemes in Multiple Disk Systems.” *IEEE Transactions on Parallel and Distributed Systems*, **6**(2), 1995.
- [Joh] C. Johnson. “US eCommerce: 2005 To 2010. <http://www.forrester.com/Research/Document/Excerpt/~~~~~0,7211,37626,00.html> (dated Sept. 2005, visited Sept. 2005).”
- [JS03] B. J. Jansen and A. Spink. “An Analysis of Web Documents Retrieved and Viewed.” In *International Conference on Internet Computing*, 2003.
- [KB94] N. Krishnakumar and A. Bernstein. “Bounded Ignorance: A Technique for Increasing Concurrency in a Replicated System.” *ACM TODS*, **19**(4), December 1994.
- [Kle99] J. Kleinberg. “Authoritative sources in a hyperlinked environment.” *Journal of the ACM*, **46**(5):604–632, September 1999.
- [KRR00] R. Kumar, P. Raghavan, S. Rajagopalan, D. Sivakumar, A. Tomkins, and E. Upfal. “Stochastic Models for the Web Graph.” In *IEEE Symposium on Foundations of Computer Science (FOCS)*, 2000.
- [Lex] LexisNexis. “LexisNexis, law, public records, company data, government, academic and business news sources <http://www.lexisnexis.com>.”
- [LG98] S. Lawrence and C. L. Giles. “Searching the World Wide Web.” *Science*, **280**(5360):98—100, 1998.
- [Lib] “Library of Congress,<http://www.loc.gov>.”
- [LLC04] V. Z. Liu, R. C. Luo, J. Cho, and W. W. Chu. “DPro: A Probabilistic Approach for Hidden Web Database Selection Using Dynamic Probing.” In *ICDE*, 2004.
- [LLS94] R. Ladin, B. Liskov, L. Shrira, and S. Ghemawat. “Providing High Availability Using Lazy Replication.” *ACM Transactions on Computer Systems*, November 1994.

- [LM00] R. Lempel and S. Moran. “The stochastic approach for link-structure analysis (SALSA) and the TKC effect.” *Computer Networks (Amsterdam, Netherlands: 1999)*, **33**(1–6), 2000.
- [LM03] R. Lempel and S. Moran. “Predictive caching and prefetching of query results in search engines.” In *Proceedings of the International World-Wide Web Conference*, 2003.
- [LM04] R. Lempel and S. Moran. “Optimizing result prefetching in web search engines with segmented indices.” *ACM Trans. Inter. Tech.*, **4**(1), 2004.
- [LMZ01] X. Liu, K. Maly, M. Zubair, and M. L. Nelson. “Arc - An OAI Service Provider for Digital Library Federation.” *D-Lib Magazine*, **7**(4), 2001.
- [LMZ02] X. Liu, K. Maly, M. Zubair, and M. L. Nelson. “DP9-An OAI Gateway Service for Web Crawlers.” In *IEEE and ACM Joint Conference on Digital Libraries (JCDL)*, 2002.
- [Loo] “Looksmart Inc. <http://www.looksmart.com>.”.
- [LS01] C. Lagoze and H. V. Sompel. “The Open Archives Initiative: Building a low-barrier interoperability framework.” In *IEEE and ACM Joint Conference on Digital Libraries (JCDL)*, 2001.
- [LS03] X. Long and T. Suel. “Optimized Query Execution in Large Search Engines with Global Page Ordering.” In *Proceedings of the International Conference on Very Large Databases (VLDB)*, 2003.
- [LS05] X. Long and T. Suel. “Three-level caching for efficient query processing in large Web search engines.” In *Proceedings of the International World-Wide Web Conference*, pp. 257–266, New York, NY, USA, 2005. ACM Press.
- [LWP01] L. Lim, M. Wang, S. Padmanabhan, J. S. Vitter, and R. C. Agarwal. “Characterizing Web Document Change.” In *Proceedings of the Second International Conference on Advances in Web-Age Information Management*, pp. 133–144. Springer-Verlag, 2001.
- [LWP03] L. Lim, M. Wang, S. Padmanabhan, J. S. Vitter, and R. Agarwal. “Dynamic Maintenance of Web Indexes Using Landmarks.” In *Proceedings of the Twelfth WWW Conference*, Budapest, Hungary, 2003.
- [Man] B. B. Mandelbrot. *Fractal Geometry of Nature*. W. H. Freeman & Co.

- [Mar01] E. P. Markatos. “On caching search engine query results.” *Computer Communications*, **24**(2), 2001.
- [MCL05] G. Mishne, D. Carmel, and R. Lempel. “Blocking Blog Spam with Language Model Disagreement.” In *1st International Workshop on Adversarial Information Retrieval on the Web*, 2005.
- [MD05] P. Metaxas and J. DeStefano. “Web Spam, Propaganda and Trust.” In *1st International Workshop on Adversarial Information Retrieval on the Web*, 2005.
- [MM00] B. H. Murray and A. Moore. “Sizing the Internet.” White paper, Cyveillance, Inc., 2000.
- [Moe92] W. Moen. “The ANSI/NISO Z39.50 Protocol: Information Retrieval in the Information Infrastructure, <http://www.cni.org/pub/NISO/docs/Z39.50-brochure/50.brochure.toc.html>,” 1992.
- [MRY01] S. Melnik, S. Raghavan, B. Yang, and H. Garcia-Molina. “Building a distributed full-text index for the web.” *ACM Trans. Inf. Syst.*, **19**(3):217–241, 2001.
- [MS99] C. D. Manning and H. Schutze. *Foundations of Statistical Natural Language Processing*. The MIT Press, 1999.
- [MSN] “MSN Search.” <http://search.msn.com>.
- [NCC05] A. Ntoulas, G. Chao, and J. Cho. “The Infocious Web Search Engine: Improving Web Searching through Linguistic Analysis.” In *Proceedings of the International World-Wide Web Conference*, 2005.
- [NCO04] A. Ntoulas, J. Cho, and C. Olston. “What’s new on the web?: the evolution of the web from a search engine perspective.” In *Proceedings of the International World-Wide Web Conference*, 2004.
- [New04] New York Times. “Old Search Engine, the Library, Tries to Fit Into a Google World. Available at: <http://www.nytimes.com/2004/06/21/technology/21LIBR.html>. Last accessed: 06/30/2006.”, June 2004.
- [Nie] J. Nielsen. “Statistics for Traffic Referred by Search Engines and Navigation Directories to Useit <http://useit.com/about/searchreferrals.html> (dated April 2004, visited Sept. 2005).”.

- [NNM06] A. Ntoulas, M. Najork, M. Manasse, and D. Fetterly. "Detecting Spam Web Pages Through Content Analysis." In *Proceedings of the International World-Wide Web Conference*, 2006.
- [NW01] M. Najork and J. L. Wiener. "Breadth-first crawling yields high-quality pages." In *Proceedings of the International World-Wide Web Conference*, 2001.
- [NZC04] A. Ntoulas, P. Zerfos, and J. Cho. "Downloading Hidden Web Content." Technical report, UCLA, 2004.
- [oclc] "Online Computer Library Center <http://wcp.oclc.org>".
- [odp] "Open Directory Project <http://www.dmoz.org>".
- [Ols02] S. Olsen. "Does search engine's power threaten Web's independence?" <http://news.com.com/2009-1023-963618.html>, 2002.
- [OW02] C. Olston and J. Widom. "Best-Effort Cache Synchronization with Source Cooperation." In *Proceedings of the International Conference on Management of Data (SIGMOD)*, May 2002.
- [PBL00] A. Paepcke, R. Brandriff, R. Larson, B. Ludaescher, S. Melnik, and S. Raghavan. "Search Middleware and the Simple Digital Library Interoperability Protocol." *D-Lib Magazine*, 6(3), 2000.
- [PBM99] L. Page, S. Brin, R. Motwani, and T. Winograd. "The PageRank Citation Ranking: Bringing Order to the Web." Technical Report 1999-66, Database Group, Computer Science Department, Stanford University, November 1999. <http://dbpubs.stanford.edu/pub/1999-66>.
- [Per01] A. Perkins. "The Classification of Search Engine Spam.", 2001.
- [PP97] J. Pitkow and P. Pirolli. "Life, Death, and Lawfulness on the Electronic Frontier." In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, Atlanta, Georgia, 1997.
- [Pub] "PubMed Medical Library, <http://www.pubmed.org>".
- [PZS96] M. Persin, J. Zobel, and R. Sacks-Davis. "Filtered Document Retrieval with Frequency-Sorted Indexes." *Journal of the American Society of Information Science*, 47(10), 1996.
- [Qui93] J.R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan-Kaufman, 1993.

- [Qui96] J. R. Quinlan. “Bagging, Boosting, and C4.5.” In *13th National Conference on Artificial Intelligence and 8th Innovative Applications of Artificial Intelligence Conference*, 1996.
- [RD02] M. Richardson and P. Domingos. “The Intelligent Surfer: Probabilistic Combination of Link and Content Information in PageRank.” In *Advances in Neural Information Processing Systems*, 2002.
- [RG01] S. Raghavan and H. Garcia-Molina. “Crawling the Hidden Web.” In *Proceedings of the International Conference on Very Large Databases (VLDB)*, 2001.
- [SDH98] M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz. “A Bayesian Approach to Filtering Junk E-Mail.” Technical report, 1998.
- [SM83] G. Salton and M. J. McGill. *Introduction to modern information retrieval*. McGraw-Hill, first edition, 1983.
- [SMZ01] P. C. Saraiva, E. S. de Moura, N. Ziviani, W. Meira, R. Fonseca, and B. Riberio-Neto. “Rank-preserving two-level caching for scalable search engines.” In *SIGIR '01: Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, 2001.
- [TG93] A. Tomasic and H. Garcia-Molina. “Performance of Inverted Indices in Shared-Nothing Distributed Text Document Information Retrieval Systems.” In *Proceedings of 2nd International Conference on Parallel and Distributed Information Systems*, 1993.
- [USP] “United States Patent and Trademarks Office.” <http://www.uspto.gov>.
- [WAA01] Y.-L. Wu, D. Agrawal, and A. El Abbadi. “Using the Golden Rule of Sampling for Query Estimation.” In *Proceedings of the International Conference on Management of Data (SIGMOD)*, 2001.
- [WD05] B. Wu and B. Davison. “Identifying Link Farm Spam Pages.” In *Proceedings of the International World-Wide Web Conference*, 2005.
- [Web] “WebArchive Project.” <http://webarchive.cs.ucla.edu>.
- [WL93] W. Y. P. Wong and D. L. Lee. “Implementations of Partial Document Ranking Using Inverted Files.” *Information Processing and Management*, **29**(5), 1993.

- [WMB99] I. H. Witten, A. Moffat, and T. C. Bell. *Managing gigabytes (2nd ed.): compressing and indexing documents and images*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.
- [WMS97] D. D. Wackerly, W. Mendenhall, and R. L. Scheaffer. *Mathematical Statistics With Applications*. PWS Publishing, 1997.
- [XO02] Y. Xie and D. O'Hallaron. "Locality in Search Engine Queries and Its Implications for Caching." In *Infocom*, 2002.
- [ZGG04] H. Zhang, A. Goel, R. Govindan, K. Mason, and B. Van Roy. "Making Eigenvector-Based Systems Robust to Collusion." In *3rd International Workshop on Algorithms and Models for the Web Graph*, 2004.
- [Zip49] G. K. Zipf. *Human Behaviour and the Principle of Least Effort: an Introduction to Human Ecology*. Addison-Wesley, first edition, 1949.
- [ZL94] Q. Zhu and P.-A. Larson. "A Query Sampling Method of Estimating Local Cost Parameters in a Multidatabase System." In *Proceedings of the International Conference on Data Engineering (ICDE)*, 1994.