Models and Methods for Web Archive Crawling

Dissertation
zur Erlangung des Grades
Doktor der Ingenieurwissenschaften (Dr.-Ing.)
der Naturwissenschaftlich-Technischen Fakultät I
der Universität des Saarlandes

Dimitar Denev
Max-Planck-Institut für Informatik

Saarbrücken
2012

Dekan der Naturwissenschaftlich-Technischen Fakultät I	Univ.-Prof. Mark Groves
Vorsitzender der Prüfungskommission	Prof. Dr. Manfred Pinkal
Berichterstatter	Prof. Dr.-Ing. Gerhard Weikum
Berichterstatter	Prof. Dr. Ralf Schenkel
Beisitzer	Dr.-Ing. Fabian Suchanek
Tag des Promotionskollquiums	20.08.2012

Eidesstattliche Versicherung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe.

Die aus anderen Quellen oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet.

Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder ähnlicher Form in einem Verfahren zur Erlangung eines akademischen Grades vorgelegt.

(Dimitar Denev)

To my parents, Stanka and Velin Denevi

Acknowledgment

I would like to express my sincere gratitude to my advisor, Prof. Dr.-Ing. Gerhard Weikum. This work would not have been possible without his steadfast support. I would like to thank him for his scientific guidance, insightful comments, exemplary leadership. His optimistic attitude, unwavering encouragement, and good humour kept my spirits up in every moment during the past four years. I would like to also thank Dr. Ralf Schenkel for the kindness he always showed to me and for accepting my request to review the thesis.

I am deeply indebted to my co-advisors Dr. Marc Spaniol and Dr. Arturas Mazeika. Marc earned my high appreciation with his creative ideas and readiness always to lend me a helping hand. Arturas was involved in most of the details of the work. His ambition and determination were key factors for the successful publication of our joint research.

Finally, I thank all my friends in MPI and Saarbrücken (especially my officemates Bilyana and Shady) for the stimulating discussions and the memorable moments we enjoyed together.

Abstract

Web archives offer a rich and plentiful source of information to researchers, analysts, and legal experts. For this purpose, they gather Web sites as the sites change over time. In order to keep up to high standards of data quality, Web archives have to collect all versions of the Web sites. Due to limited resuources and technical constraints this is not possible. Therefore, Web archives consist of versions archived at various time points without guarantee for mutual consistency.

This thesis presents a model for assessing the data quality in Web archives as well as a family of crawling strategies yielding high-quality captures. We distinguish between single-visit crawling strategies for exploratory and visit-revisit crawling strategies for evidentiary purposes. Single-visit strategies download every page exactly once aiming for an "undistorted" capture of the ever-changing Web. We express the quality of such the resulting capture with the "blur" quality measure. In contrast, visit-revisit strategies download every page twice. The initial downloads of all pages form the visit phase of the crawling strategy. The second downloads are grouped together in the revisit phase. These two phases enable us to check which pages changed during the crawling process. Thus, we can identify the pages that are consistent with each other. The quality of the visit-revisit captures is expressed by the "coherence" measure.

Quality-conscious strategies are based on predictions of the change behaviour of individual pages. We model the Web site dynamics by Poisson processes with page-specific change rates. Furthermore, we show that these rates can be statistically predicted. Finally, we propose visualization techniques for exploring the quality of the resulting Web archives.

A fully functional prototype demonstrates the practical viability of our approach.

Kurzfassung

Ein Webarchiv ist eine umfassende Informationsquelle für eine Vielzahl von Anwendern, wie etwa Forscher, Analysten und Juristen. Zu diesem Zweck enthält es Repliken von Webseiten, die sich typischerweise im Laufe der Zeit geändert haben. Um ein möglichst umfassendes und qualitativ hochwertiges Archiv zu erhalten, sollten daher - im Idealfall - alle Versionen der Webseiten archiviert worden sein. Dies ist allerdings sowohl aufgrund mangelnder Ressourcen als auch technischer Rahmenbedingungen nicht einmal annähernd möglich. Das Archiv besteht daher aus zahlreichen zu unterschiedlichen Zeitpunkten erstellten "Mosaiksteinen", die mehr oder minder gut zueinander passen.

Diese Dissertation führt ein Modell zur Beurteilung der Datenqualität eines Webarchives ein und untersucht Archivierungsstrategien zur Optimierung der Datenqualität. Zu diesem Zweck wurden im Rahmen der Arbeit "Einzel-" und "Doppelarchivierungsstrategien" entwickelt. Bei der Einzelarchivierungsstrategie werden die Inhalte für jede zu erstellende Replik genau einmal gespeichert, wobei versucht wird, das Abbild des sich kontinuierlich verändernden Webs möglichst "unverzerrt" zu archivieren. Die Qualität einer solchen Einzelarchivierungsstrategie kann dabei durch den Grad der "Verzerrung" (engl. "blur") gemessen werden. Bei einer Doppelarchivierungsstrategie hingegen werden die Inhalte pro Replik genau zweimal besucht. Dazu teilt man den Archivierungsvorgang in eine "Besuchs-" und "Kontrollphase" ein. Durch die Aufteilung in die zuvor genannten Phasen ist es dann möglich festzustellen, welche Inhalte sich im Laufe des Archivierungsprozess geändert haben. Dies ermöglicht exakt festzustellen, ob und welche Inhalte zueinander passen. Die Güte einer Doppelarchivierungsstrategie wird dazu mittels der durch sie erzielten "Kohärenz" (engl. "coherence") gemessen.

Die Archivierungsstrategien basieren auf Vorhersagen über das Änderungsverhalten der zur archivierenden Inhalte, die als Poissonprozesse mit inhaltsspezifischen Änderungsraten modelliert wurden. Weiterhin wird gezeigt, dass diese Änderungsraten statistisch bestimmt werden können. Abschließend werden Visualisierungstechniken für die Qualitätsanalyse des resultierenden Webarchivs vorgestellt. Ein voll funktionsfähiger Prototyp demonstriert die Praxistauglichkeit unseres Ansatzes.

Summary

Web archives undertake the impossible-on-first-glance task to prevent information on the Web from disappearing. With the help of large-scale Web crawlers, they capture Web sites and save the snapshots.

Exploring Web archives can help sociologists, politologists, and media analysts reflect on the zeitgeist of the past decades. For example, a comprehensive Web archive of sites following election campaigns reveals the issues and the problems that societies face. Also, Web archives as evidence can be invaluable to experts on intellectual property (IP, e.g., at patent offices) and compliance with Internet legislation (e.g., for consumer services). When a company is accused of violating IP rights (regarding inventions or trademarks), it may want to prove the existence of certain phrases on its Web pages as of a certain timepoint in the past. Conversely, an Internet-fraud investigation may aim at proving the absence of certain phrases (e.g., proper pricing statements or rights of withdrawal) on a Web site. Such scenarios entail that Web archives need to be maintained with a high standard of data quality.

However, during a site crawl, pages may undergo changes which hinder both exploratory and evidentiary usages of Web archives. To understand fully this issue and develop methods for quality-conscious Web archiving, several challenging research problems must be addressed first. In this thesis, we propose solutions dealing with each of the problems as follows.

- **Web archiving model.** Currently, there are no Web archiving models which cover the data quality of Web archives. While the existing approaches focus on the organization of the physical storage and the exploration of the captured Web content, no effort until now has been made in investigating the quality of Web archives. In this thesis, we a propose a model with two quality measures — blur and coherence — that express the credibility of Web archives for exploratory and evidentiary purposes.

- **Quality-conscious crawling strategies.** The majority of Web archives employ

breadth-first crawling strategy. The user can modify only the scope of the crawl in terms of domain, topic, and depth. Strategies that optimize for freshness might also be used. The freshness metric, however, is not a reliable attestation for the exploratory or the evidentiary usage of the Web archive. We devise crawling strategies that aim to optimize our archive quality measures. For exploratory usage of the archive we suggest a family of single-visit strategies. For evidentiary usage which require deterministic quality guarantees we introduce visit-revisit crawl strategies that visit pages twice: a first visit to fetch the page and a later revisit to validate that the page has not changed.

- **Change prediction.** Crawling strategies order page visits primarily based on change prediction. The diversity of the change behaviour of Web pages hinders the development of a change model which is both universal and precise. We need to strike a balance between. To do this, we model Web site dynamics by Poisson processes with page-specific change rates. Furthermore, we show that these rates can be statistically predicted based on page types (e.g., MIME types), depths within the site (e.g., distance to site index pages), and URLs (e.g., manually edited user homepages vs. pages generated by content management systems). If we do not have enough training data, we substitute it with the change rates given by the available sitemaps.

- **Data quality exploration.** Exploring the data quality of the Web archive is essential for the continuous improvement of the Web archive. Visualizations of the changed pages help archivists modify the scope of future crawls in a way that increases the quality of the capture. In spite the existence of many techniques for graphical analytics and the development of user interfaces to Web archives, nothing until now has been done to bring them together for the purpose of Web archive quality. To address this, we present visualization techniques for exploring the quality of the Web archives. With the help of area plots, scatter plots, and graph visualizations, the user gains insights on the data quality of the archives and the change behaviour of the Web pages. Then, she is able to fine-tune future crawls for improved quality.

Zusammenfassung

Die Webarchivierung unternimmt den auf den ersten Blick unmöglichen Versuch Inhalte des Web vor dem Verschwinden zu bewahren, indem sie periodisch Repliken von Webinhalten erstellt. Ein umfassendes Webarchiv spiegelt somit den Zeitgeist einer gesamten Epoche wider und ermöglicht etwa Soziologen, Politologen oder Medienanalysten umfassende Studien. So lassen sich etwa Aussagen politischer Parteien über die Zeit hinweg untersuchen und kontrastieren. Ferner kann ein Webarchiv einen unschätzbaren Wert in Patentfragen oder bei der Verletzung von geistigen Eigentumsrechten darstellen. Dazu ist es bei einem Rechtsstreit zumeist erforderlich die Verletzung rechtlicher Vorgaben (z.B. ungültige AGBs, Plagiarismusvorwürfe oder urheberrechtsverletztende Inhalte) durch eine Replik einer bestimmten Webseite zu einem konkreten Zeitpunkt nachzuweisen. Diese Anforderungen führen dazu, dass Webarchive mit hohen Qualitätsstandards erforderlich sind.

Besonders problematisch für die Qualität eines Archivs ist jedoch, dass sich Webinhalte während des Archivierungsvorgangs verändern können. Aus diesem Grund leistet diese Arbeit die folgenden Beiträge zur Steigerung der Datenqualität in Webarchiven:

- **Webarchivierungsmodell.** Derzeit gibt es kein Modell, das die Datenqualität eines Webarchivs beschreibt. Während der Fokus bestehender Ansätze auf der Organisation des physikalischen Speichers lag, wurden keine Anstrengungen unternommen die eigentliche Qualität des Webarchivs zu untersuchen. In dieser Arbeit wird daher ein Modell mit zwei Qualitätsmaßen, “Verzerrung” (engl. “blur”) und “Kohärenz” (engl. “coherence”), eingeführt.

- **Qualitätsorientierte Archivierungstrategien.** Der Standardansatz zur Webarchivierung beruht auf einer der Breitensuche verwandten Archivierungsstrategie. Dieser Ansatz führt jedoch nicht dazu, ein möglichst “unverzerrtes” oder gar “kohärentes” Webarchiv zu erhalten. Daher werden “Einzel-” und “Doppelarchivierungsstrategien” zur Erstellung einzelner Repliken des Archivs eingeführt. Die Einzelarchivierungsstrate-

gie dient dazu die “Verzerrung” zu minimieren. Dahingegen wird die Doppelarchivierungsstrategie dazu verwendet die “Kohärenz” zu maximieren.

- **Änderungsvorhersage.** Die zur Qualitätsoptimierung entwicklten Archivierungsstrategien basieren auf Vorhersagen über das Änderungsverhalten der zur archivierenden Inhalte. Die Vorhersage dieser Änderungen ist von vielen äußeren Faktoren beeinflusst und nur schwer exakt zu bestimmen. Dennoch lassen sich gute Annäherungen durch die Modellierung als Poissonprozess mit inhaltsspezifischen Änderungsraten erzielen. Diese beruhen u.a. auf dem MIME-Typ, der Tiefe der Seite im Bezug auf die Indexseite und Struktur der URLs (z.B. manuell oder automatisch erstellt).
- **Visuelle Datenqualitätsuntersuchung.** Die Bewertung der Datenqualität eines Webarchives ist unabdingbar um eine kontinuierliche Verbesserung des Archives zu erzielen. Die Visualisierung des Änderungsverhaltens von Webinhalten unterstützt dabei den Archivar beim Planen zukünftiger Archivierungsprozesse. Zu diesem Zweck werden Visualisierungstechniken in Form von Area-Plots, Scatter-Plots und Graphvisualisierungen präsentiert.

Contents

Chapter 1.

Introduction

1.1. Archives and Web Archiving

Throughout history, collective memory has been the backbone of cultural and societal development. With the advance of technology, it has taken various forms: from inscriptions and papyrus to printed books and video records. These media preserved the accumulated knowledge and transmitted it to the next generations. Archives and libraries ensured that artefacts with high cultural and evidentiary value are safeguarded against the will of time.

Nowadays, archives and libraries face new challenges in their endeavour to preserve knowledge in the long term. They undertake the impossible-on-first-glance task to prevent information on the Web from disappearing. With the help of large-scale Web crawlers, they capture Web sites and save the snapshots in Web archives. National libraries (e.g., `www.loc.gov`, `www.webarchive.org.uk`, `netarkivet.dk`, `www.bnf.fr`, `www.webarchiv.cz`, etc) and organizations like the Internet Archive (`archive.org`) and the European Archive (`europarchive.org`) take decisive steps in this direction. At the same time they must guarantee that Web archives can serve the same purposes as traditional archives do, namely exploratory and evidentiary purposes.

Exploring Web archives can help sociologists, politologists, and media analysts reflect on the zeitgeist of the past decades. For example, a comprehensive Web archive of sites following election campaigns reveals the issues and the problems that societies face. One can draw conclusions about the existing attitudes and the proposed solutions. By looking back at the choices taken in the past, one sees their positive and negative effects in the present. Thus, people can make informed decisions what policy and which politicians to support.

Also, Web archives as evidence can be invaluable to experts on intellectual property

(IP, e.g., at patent offices) and compliance with Internet legislation (e.g., for consumer services). For example, when a company is accused of violating IP rights (regarding inventions or trademarks), it may want to prove the existence of certain phrases on its Web pages as of a certain timepoint in the past. Conversely, an Internet-fraud investigation may aim at proving the absence of certain phrases (e.g., proper pricing statements or rights of withdrawal) on a Web site. Clearly, these scenarios entail that Web archives need to be maintained with a high standard of data quality.

1.2. Quality of Web archives

One can argue that the quality of the Web archive depends on the selected Web sites or the frequency of Web site revisits. Clearly, high-quality Web sites and frequent crawls improve the overall archive quality. These decisions, however, are up to the archive curators and no matter what they decide, insufficient quality inherent from the existing Web crawling approaches will have negative impact.

Web crawling practice. Research in Web crawling received a boost from the rapid development of search engines. Web crawling is essential for search engines to build an index of the Web. However, crawling for archiving substantially differs from crawling performed by major search providers. Search engines aim at broad coverage of the Web, target the most important pages, and schedule revisits of the pages based on their individual freshness and importance. It may even be sufficient to see the `href` anchor in order to index a page without ever visiting the page itself. In contrast, archive curators are interested in a complete capture of a site either at reasonably regular time points (weekly, monthly, quarterly) or in aftermaths (natural disasters, political scandals, research projects).

Data quality issues. During a site crawl, pages may undergo changes which hinder both the exploratory and evidentiary usage of Web archives. Changes during crawling result in a blurred snapshot of the site. We borrow the terms *blur* from photography to denote the quality of the snapshot. Similarly to photography, the longer the exposure time (timespan of the entire site crawl), the higher the risk of blurring the capture (archiving pages in different states of the site). In contrast, if the site's pages did not change at all (or changed insignificantly) during the crawl, we say that the pages are sharp and the

snapshot is coherent (or almost coherent). Avoiding blurred captures is important for the quality assurance of the Web archive and its professional usage. Ideally, a user should get mutually consistent pages. In case mutual consistency of pages cannot be fully assured, there should at least be guarantees about data quality.

Data quality for exploratory usage. Consider an analyst who studies a politician's behavior and success during an election campaign, based on weekly or daily crawls of the corresponding party's Web site (which may include user-provided contents in associated wikis or blogs). Suppose one page of the site covers a television debate with the politician, pointing to other pages with "opinion barometers" for the politician and her opponents in the debate. Each of these "barometer pages" in turn points to recent public appearances of the featured politician. As these pages frequently change, the archived snapshot contains page versions as of different timepoints. Now, already an incoherence by a few hours difference between the captures of these interrelated pages can lead to misinterpretations and wrong conclusions by the analyst. For example, the analyst may see a brilliant performance of the politician on the debate page, then follow the pointer to a barometer page with unfavorable public opinions, simply because the barometer was captured earlier.

Data quality for evidentiary usage. As a real-life case in point, an archive of a Web site was disapproved as evidence in a lawsuit about intellectual property rights [80] because the judge considered the archive as having insufficient quality and no guarantees about the consistency of its content. In such cases, a strategy for getting coherent site captures or precisely stating the level of consistency would make a big difference.

Naïve crawling strategy. The simplest strategy to obtain a coherent capture of a Web site and avoid anomalies would be to freeze the entire site during the crawl period. This naïve approach is impractical as an external crawler cannot prevent the site from posting new information on its pages or changing its link structure. On the contrary, the politeness etiquette for Internet robots forces the crawler to pause between subsequent HTTP requests, so that the entire capturing of a medium-sized site (e.g., a university) may take many hours or several days. Long crawl duration is an issue for search-engine crawlers, too, but it is more severe for archive crawlers as they cannot stop a breadth-first site exploration once they have seen enough `href` anchors. So, slow but complete site crawls drastically increase the risk of blurred captures.

Alternative naïve crawling strategy. An alternative strategy that may come to mind would be to repeat a crawl that fails to yield a coherent capture and keep repeating it until eventually a blur-free snapshot can be obtained. But these repetitions are an unacceptably high price for data quality as the crawler operates with limited resources (servers and network bandwidth) and needs to carefully assign these to as many different Web sites as possible.

Sitemaps. Website masters can help to make archiving easier by providing additional information about the site: its structure, its typical change patterns, hints about where and when changes occur, and so on. The recently introduced *sitemaps* protocol [85] can provide a list of URLs and metadata about page (or sub-directory-level) modifications so that crawlers can more intelligently process the site. However, sitemaps alone are merely hints that can guide a crawl strategy (e.g., towards pages with high likelihood of having changed so that, for example, a search-engine robot should obtain a fresh version). Sitemaps are not a solution for ensuring data quality.

1.3. Research Challenges

While the issue of Web-archive quality is obvious, it is unclear how to formalize the problem and address it technically. In order to understand completely this issue, we must address the challenges we list next.

Web archiving model. Currently, there are no Web archiving models which cover the data quality of the Web archives. While the existing approaches focus on the organization of the physical storage and the exploration of the captured Web content, no effort has been made in defining quality measures. The relation between the structure of the Web archive and its credibilty as a source of information is not yet explored.

Quality-conscious crawling strategies. The lack of quality metrics results in lack of quality-conscious crawling strategies. Instead, the Web archives employ breadth-first crawling strategy. The user can modify only the scope of the crawl in terms of domain, topic, and depth. Strategies that optimze for freshness might also be used. The freshness metric, however, is not a reliable attestation for the exploratory or the evidentiary usage of the Web archive. Novel strategies optimizing for archive quality are necessary.

Change prediction. Crawling strategies order page visits primarily based on change prediction. The diversity of the change behaviour of Web pages hinders the development of a change model which is both universal and precise. We need to strike a balance. On one hand, any quality metric, which depends on page changes, is expressed mathematically using a universal change model with few parameters. On the other hand, any crawling strategy must take into account specific change behaviours like periodicities or nearly continuous changes in a Web site, so that the resulting capture is of high quality.

Data quality exploration. Exploring the data quality of the Web archive is essential for the continuous improvement of the Web archive. Visualizations of the changed pages help the archivists to modify the scope of the future crawls in a way which increases the quality of the capture. In spite the existence of many techniques for graphical analytics and the development of user interfaces to Web archives, nothing until now has been done to bring them together for the purpose of Web archive quality.

1.4. Contributions

In this thesis, we address the points highlighted in the previous section and present a framework for quality assurance of Web archives, coined SHARC for Sharp Archiving of Web-Site Captures. It consists of a model of quality properties as well as a suite of algorithms for crawling that allow us to assess and optimize site-capturing strategies. We summarize the contributions as follows:

1. **Quality Metrics:** Our SHARC framework introduces two measures of data quality for site captures:
   - *Blur* is a stochastic notion that reflects the expected number of page changes that a time-travel access to a site capture would accidentally see, instead of the ideal view of a instantaneously captured, "sharp" site. We assume that the timepoints to which analysts will later refer for snapshot analysis are uniformly distributed over time, hence the stochastic approach. The blur quality measure was published in the proceedings of the 35th International Conference on Very Large Data Bases (VLDB '10) [33].
   - *Coherence* is a deterministic quality measure that counts the number of unchanged and thus coherently captured pages in a site snapshot. Here "unchanged" denotes pages that are definitely known to be invariant throughout

some time window, ideally the entire crawl. The setting allows us to guarantee mutual consistency across several pages in a snapshot that are logically interrelated (e.g., a television debate and the "barometer" pages about the participating politicians). The coherence quality measure was published first in the proceedings of the 3rd Workshop on Information Credibility on the Web (WICOW '09) [92] and further developed in [33, 34].

2. **Crawling Strategies:** We devise crawl strategies that aim to optimize our archive quality measures. While stochastic guarantees like blur are good enough for explorative use of the archive (while keeping crawl costs low), access that aims to prove or disprove claims about interrelated contents in site snapshots needs deterministic guarantees like coherence and would accept higher crawl costs. For explorative use of the archive it is sufficient to visit each page once. The order of the downloads is a degree of freedom for the crawl scheduler. For deterministic guarantees we introduce crawl strategies that visit pages twice: a first visit to fetch the page and a later revisit to validate that the page has not changed. The order of visiting and revisiting pages is a degree of freedom for the crawl scheduler. For very large Web sites, it is unrealistic to obtain a coherent capture for the entire site. In this case, we opt for smaller subsites of interrelated pages and derive these via the sitemaps protocol. SHARC provides a suite of novel algorithms for archive crawling:

   - *SHARC-Offline* assumes a-priori knowledge of all URLs and their specific change rates, and arranges downloads in an organ-pipe manner with the hottest pages in the middle. It minimizes blur and provides stochastic guarantees about data quality.
   - *SHARC-Online* drops these assumptions and operates with an estimate of the number of pages on the site but without prior knowledge of any URLs other than the crawl's entry point. The algorithm aims to approximate the organ-pipe shape, but can lead to suboptimal schedules.
   - *SHARC-Revisits* visits pages twice aiming to minimize the stochastic blur and allowing change detection during the crawl.
   - *SHARC-Intervals*, similarly to SHARC-Revisits, visits pages twice aiming to minimize the coherence. The algorithm computes a confidence interval for each page and schedules the visits and the revisits in way that the visit-revisit

interval of a page does not exceed the corresponding confidence interval.

- *SHARC-Threshold* is visit-revisit strategy alternative to SHARC-Intervals. The algorithm uses change rates of Web pages to arranges the visits and revisits. A user-specified parameter helps the algorithm determine continuously changing pages (very hot pages) and adjusts the scheduling of visits and revisits so that the other, not so hot, pages have a higher chance of getting coherently captured.
- *SHARC-Selective* is an improvement over SHARC-Threshold. It automatically detects very hot pages and updates the download schedule.

The strategies that minimize the blur (SHARC-Offline, SHARC-Online, SHARC-Revisits) were introduced in the already mentioned publication [33]. The SHARC-Intervals strategy was published in the proceedings of the 10th International Web Archiving Workshop (IWAW '10) [61] under the name SOLAR. The SHARC-Selective was described in the special issue of VLDB Journal "Best papers of VLDB 2009" in April 2011 [34]. This strategy is an improved version of SHARC-Threshold described in [92].

3. **Change Predictions:** The SHARC framework needs estimates for the frequencies at which changes occur in a Web site. In line with the prior literature [3, 26, 75], we model site changes by Poisson processes with page-specific change rates. We show that these rates can be statistically predicted based on page types (e.g., MIME types), depths within the site (e.g., distance to site index pages), and URLs (e.g., manually edited user homepages vs. pages generated by content management systems). If we do not have enough training data, we substitute it with the change rates given by the available sitemaps. We presented evaluation of our predictors in [34].

4. **Integrated Prototype:** We designed and implemented a prototype of the SHARC framework based on the Heritrix Web crawler [65]. The source code of the prototype is available online in the source repository of the Living Web Archives project [1]. The prototype has been used in practice for Web archiving by the partners in the project.

[1] http://code.google.com/p/liwa-technologies/

5. **Visual Analysis of Data Quality:** We present visualization techniques for exploring the quality of the Web archives. With the help of area plots, scatter plots graph visualization, the user gains insights on the data quality of the archives and the change behaviour of the Web pages. This helps for fine-tuning future crawls for improved quality. We published our initial findings in the proceedings of the 9th International Web Archiving Workshop (IWAW '09) [93].

Our experimental studies are carried out with both synthetically generated Web sites and repeated crawls of different-sized domains including `mpi-inf.mpg.de` (MPII), `dmoz.org` (DMOZ), and sites from the .uk.gov collection (UKGOV). For change prediction we use standard machine learning algorithms such as Naive-Bayes and C4.5 classifiers. The experiments demonstrate the practical viability of our approach and the advantages of our algorithms compared to more traditional crawl strategies.

1.5. Thesis Outline

The thesis is organized as follows. Chapter 2 reviews related work and the state of the art in Web archiving. Chapter 3 introduces our computational model for Web archiving and site capturing. Chapters 4 and 5 present our single-visit and visit-revisit strategies. Chapter 6 describes our approach for change prediction. Chapter 7 presents our system architecture and a prototype implementation based on the open-source archive crawler Heritrix. Chapter 8 discusses visualization techniques for exploring and analyzing data quality of Web archives. We conclude the thesis with a summary and highlights of future research directions in Chapter 9.

Chapter 2.

Related Work

In this chapter we present existing models in topics related to the thesis. We start with Web archiving, then focus on the dynamics of the Web, proceed with Web crawling and finish with the visualization of changes in Web archives.

2.1. Web Archiving

Masanès [59] presents a systematic view on the current issues in Web archiving. Three of them are closely related to our work: ranking mechanisms for Web pages, the scale of the Web, and guidelines for crawling that prevent possible conflict between Web site owners and Web archives.

Ranking Mechanisms

A ranking mechanism is essential for setting priorities to Web pages during archiving, especially, when the archiving institution has limitations on its resources (bandwidth, physical storage, or time). Some solutions come from the Web Search field where estimating page importance is crucial. The estimations may be based on the Webgraph, the page content, the user behaviour or any combination of the above.

The Webgraph is a directed graph representing the World Wide Web. The nodes of the graph correspond to all the pages in the Web. Pages p_1 and p_2 are connected by a directed edge if there is a hyperlink from p_1 to $p2$. PageRank [15] and HITS [52] algorithms are the first algorithms to analyze the Webgraph structure for page importance.

PageRank is a recursive algorithm that assigns a weight to a page p based on the number and the weights of the pages pointing to p. The weight corresponds to the probability that a "random surfer" reaches p. A random surfer is a person who follows at random an

outgoing link every time she visits a page. Mathematically, this can be expressed by the following formula:

$$PageRank(p) = \sum_{p' \in In(p)} \frac{PageRank(p')}{|Out(p')|},$$

where p and p' are pages, $PageRank(p)$ is the PageRank value of p, $In(p)$ is the set of incoming links to p, and $Out(p)$ is the set of outgoing links from p. The formula does not take into account that at each step the person may follow an outgoing link or jump at a random page by typing the URL or using her bookmarks. To accommodate for that, the formula is modified by introducing a damping factor d which is the probability that the person follows an outgoing link. The modified formula is below:

$$PageRank(p) = \frac{1-d}{N} + d \sum_{p' \in In(p)} \frac{PageRank(p')}{|Out(p')|},$$

where N is the total number of Web pages and d is the damping factor. It is typically assumed that $d = 0.85$ [15].

When used in practice, PageRank often results in a "rich-get-richer" phenomenon: popular pages become even more popular at expense of the less popular pages. This is a problem for new pages which do not get enough visibility. Cho et al. [28] address the problem by introducing the measure *page quality* Q. They express the page quality of a page p as a function of the PageRank and its increase in time:

$$Q(p,t_1) = C\frac{PageRank(p,t_1) - PageRank(p,t_0)}{(t_1 - t_0)PageRank(p,t_1)} + PageRank(p,t_1),$$

where t_0 and t_1 are sequential time points, C is a constant reflecting the popularity of the page among the Web users.

An alternative to PageRank is the Hyperlink-Induced Topic Search (HITS) algorithm [52], commonly known as Hubs and Authorities algorithm. The algorithm is based on the observation during the early years of Internet that some Web pages (hubs) served as directories for pages with authoritative information (authorities). A good hub is a page pointing to many authoritative pages and conversely, a good authority is a page pointed by many hubs.

The algorithm assigns two values to a page p: a hub score $Hub(p)$ and an authority score $Auth(p)$. The initial values are $Hub(p) = 1$ and $Auth(p) = 1, \forall p$. The algorithm

iteratively updates the scores until convergence:

$$\begin{aligned} Hub(p) &= \sum_{p' \in Out(p)} Auth(p') \\ Auth(p) &= \sum_{p' \in In(p)} Hub(p'). \end{aligned}$$

The scores are then normalized to ensure convergence.

The aforementioned approaches use the Webgraph to estimate page importance. Although important, the Webgraph is not the only source for that. User behaviour also gives insights which pages are important and which pages are not.

Liu et al. [58] estimate page importance from users' browsing history and the time spent on individual pages. The authors build a directed graph whose edges are the transitions that took place. Then, they use the graph for a basis of a continuous-time Markov process. Finally, they compute the importance of the pages as the stationary probability distribution of the Markov process.

Similarly to [28], Dong et al. [36] aim at ranking the most recent pages on top. They use a learning-to-rank algorithm (GBrank [101]) in combination with regular ranking models. Features divided in four classes capture the recency of Web pages:

- **timestamp features**: count of timestamps, first timestamp, minimal timestamp, maximal timestamp, mean timestamp, standard deviation;
- **linktime features**: count of links, earliest link, latest link, mean linktime, standard deviation of the link times;
- **WebBuzz features**: web buzz intensity and time of the buzz;
- **page classification features**: news page confidence, blog page confidence, page quality confidence.

The combination between the recency ranking and the regular ranking may be done in two ways: either use the output score of the regular ranking as an additional feature, or learn over the complete training data of the regular ranking.

In another study, Bendersky et al. [13] rank the Web pages according to their readability. They express the readability of the page with a list of features grouped according to three factors:

- **content clarity and presentation**: number of terms on the page, average length of the terms, fraction of anchor text, entropy, ratio between stopwords and non-stopwords;

- **provision of useful links**: fraction of anchor text;
- **ease of navigation**: number of terms in the title of the page, depth of the URL, fraction of the text in the tables..

The features bring about a shift towards page quality in any ranking mechanism.

The previous study reflects to a large extent the textual quality of the Web page, the visual appeal of the pages is not considered. Wu et al.[98] suggest an approach which focuses on the page aesthetics. For that purpose, they break up the page into a tree of visual blocks and text nodes. Then, they model the artistic value of the Web page based on four factors, each with a corresponding list of features:

- **layout of the page**: number of layout blocks, aspect ratio, number of leaf nodes;
- **layout of the textual blocks**: number of text leaf nodes, fraction of the text nodes, character density;
- **classical visual properties**: hue, brightness, saturation, colourfulness;
- **image complexity**: ratio of the image size to the size of the whole page.

The visual properties of Web pages can also influence the Web archiving process as Ben Saad and Gançarski [81] demonstrate. They too break up the Web pages in visual blocks and schedule page downloads based on the change patterns of each visual block.

Quite often, though, due to requirements of the archiving institution, human input is necessary. This ensures the quality of an Web archive. The Archive-It service [1] from of Internet Archive is one of the most popular services of this kind. The service enables curators to select pages for archival. The curators enter domain, capture frequency and scope through a Web interface. The Archive-It service archives the designated Web sites, stores the archive collections, and provides Web interface for searching and browsing. There are as well similar commercial archive services [2,3,4,5,6].

Anand et al. [5] suggest a distributed platform for human-assisted Web archiving. They exploit the “wisdom of the crowd“ phenomenon. Assuming that Web users visit mostly

[1] http://www.archive-it.org
[2] http://aleph-archives.com/
[3] http://archivethe.net/
[4] http://www.hanzoarchives.com/
[5] http://www.iterasi.com/
[6] http://www.website-archive.com/

important pages, their goal is to archive users' browsing history. The proposed prototype includes a Web browser plugin that stores Web pages at every visit.

Web site owners may also specify the importance of their pages. The Sitemap protocol [85] gives them that possibility. Sitemaps are XML files that contain URLs pointing to other sitemaps (see Figure 2.1) or a list of URLs available at the site (see Figure 2.2).

A sitemap file consists of a list of URLs with the following metadata (cf. Figure 2.2):

- **loc** is a mandatory field indicating the URL of a Web page.
- **lastmod** is an optional field indicating the last modified date and time of the page.
- **changefreq** is an optional field indicating the typical frequency of change. Valid values include: *always, hourly, daily, weekly, monthly, yearly, never*. This information can be mapped onto (ranges of) change rates for the page-specific parameter of the Poisson-process model.
- **priority** is an optional field indicating the relative importance or weight of the page on the Web site.

Currently, approximately 35 million Web sites publish sitemaps, providing metadata for several billion URLs. Top domains using sitemaps are in .com, .net, .cn, .org, .jp, .de, .cz, .ru, .uk, .nl domains including `www.cnn.com`, `www.nytimes.com`, `www.bbc.co.uk`, `www.dw-world.de` [85].

```
<sitemapindex xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
<sitemap>
  <loc>http://www.cnn.com/sitemap_specials.xml</loc>
  <lastmod>2007-04-18T12:05:20-04:00</lastmod>
</sitemap>
<sitemap>
  <loc>http://www.cnn.com/sitemap_elections.xml</loc>
  <lastmod>2008-01-08T20:17:50-05:00</lastmod>
</sitemap>
```

Listing 2.1: Example of Sitemap with Sitemap Indexes

```
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
<url>
  <loc>http://www.dw-world.de</loc>
  <lastmod>2009-02-11</lastmod>
  <changefreq>hourly</changefreq>
  <priority>1.0</priority>
</url>
<url>
  <loc>http://www.dw-world.de/dw/0,,265,00.html</loc>
  <lastmod>2008-11-11</lastmod>
  <changefreq>hourly</changefreq>
  <priority>1.0</priority>
</url>
</urlset>
```

Listing 2.2: Example of Sitemap with URLs

Sitemaps as well as rest of the methods assume that an individual page with high importance increases the quality of the collection (Web archive or Web index).

In this thesis, the focus is not on the individual Web pages but on the Web archive as a whole. The goal is achieve mutual consistency of all pages. If this is not possible we aim at minimizing the incoherence. In this case, the most important pages are those which change most frequently. Notable exception are extremely *hot* pages that are almost continuously changing. If necessary, we can plug in any of the existing importance measures as a weighting factor in our quality metrics.

Scale of the Web

The size of the Web is the next important issue for Web archives. Here, we present the extent to which Web archives scale up.

Currently, the biggest archive of the Web is the WayBack Machine [7]. As of 2009, the WayBack Machine contained 30 billion pages amounting to 2PB with growth rate 100TB/month. [8] A quick calculation shows that its current size is at least 5.5PB with more than 70 billion pages. The crawls are supplied by Alexa Internet. Alexa crawls 1.6 GB per day, each snapshot takes approximately two months to complete, currently containing 4.5 billion pages from over 16 million sites [9]. Although these are big numbers, they are

[7] http://archive.org/web/web.php

[8] http://www.computerworld.com/s/article/9130081/Internet_Archive_to_unveil_massive_Wayback_Machine_data_center?taxonomyId=12&intsrc=kc_top&taxonomyName=hardware

[9] http://www.alexa.com/company/technology

by far surpassed by the 10^{12} unique URLs Google reported in 2008 [10]. Additionally, if we take into account the numerous popular platforms for user generated content (social networks for example), it is clear that a snapshot of the whole Web is impossible.

However, it is attainable to define a scope of the Web archive and then aim for a complete snapshot. Gomes et al. [42] report 42 Web archiving initiatives. 80% of them focus exclusively on content related to a country (Australia [68], Republic of Korea [71], Portugal [41] and many others), national region (North Carolina [6], Catalonia [69], Tasmania [72]) or an institution (UK Government [7], German parliament [19], Harvard University [55]). The remaining archives focus either on a group of countries (Latin America [73], Pacific Islands [70]) or on specific topics (human rights [54]).

All of these archives have a predefined scope and are interested in all Web page in that scope. Therefore, in the thesis, we assume that the completeness of a snapshot is required given a predefined scope.

Guidelines for Crawling

Finally, we discuss the potential conflicts between Web sites and Web archives. We identify two possible problems: first, the owner of the Web site may forbid certain Web pages from being archived and second a Web crawler may cause server overload with too frequent HTTP requests.

The first problem is addressed by the Standard for Robots Exclusion [53]. It has two standard directives:

- **User-agent** refers to the Web crawler(s) which must adhere to the provided crawling instructions, wildcard symbol (*) includes all Web crawlers.

- **Disallow** restricts the access to certain pages in the Web site.

The standard allows the Web site owners to specify in a *robots.txt* file which Web pages are allowed for crawling and which are not. Example 2.3 shows a simple robots.txt file which disallows crawling private Web pages.

```
User-agent: *
Disallow: /private/
```

Listing 2.3: Example of robots.txt

[10] http://googleblog.blogspot.de/2008/07/we-knew-web-was-big.html

An extension of the standard with an additional directive **Crawl-delay** makes it possible for Web site oweners to require a *politeness delay* between two successive HTTP requests. Example 2.4 shows a robots.txt file that requires delay of 10 seconds between two HTTP requests.

```
User-agent: *
Crawl-delay: 10
```

Listing 2.4: Example of robots.txt with Crawl Delay

If no delay is specified, the Web crawler based on its politeness policy decides how long the delay should be. Web crawlers differ in their politeness policy: some use 15 seconds [8], other 10 [25], or even 1 second [35]. The Mercator Web crawler [47] adapts its politeness delay based on the number of previous requests to a Web server. Similarly, The Heritrix Web crawler [65] has a self-adapting politeness policy with a default value for the delay. The default value is specified by the user (Web archive engineer) and is generally observed.

In our prototype implementation we adapt the politeness policy of Heritrix, since this crawler is the de facto standard for Web archiving. We set 0.5 seconds politeness delay for the Web site of our institute[11] and 3 seconds for other Web sites.

In this section we discussed importance metrics as a ranking mechanism, Web archiving initiatives as examples for dealing with the growing size of the Web, and politness guidelines as means for avoiding potential problems between Web site owners and Web archivists. Now we continue with an overview of the dynamics of the Web, more specifically the models of Web page changes.

2.2. Web Dynamics

Changes of the Web pages pose a severe problem for Web archivists. If pages change during the crawl, the Web archive degrades in quality. The problem has been identified more than a decade ago. The earliest example is from Brügger [17] from the year 2000:

> During the Olympics in Sydney in 2000, I wanted to save the website of the Danish newspaper Jyllands Posten. I began at the first level, the front page, on which I could read that the Danish badminton player Camilla Martin would play in the finals a half hour later. My computer took about an hour to

[11] http://www.mpi-inf.mpg.de

> save this first level, after which time I wanted to download the second level, "Olympics 2000". But on the front page of this section, I could already read the result of the badminton finals (she lost).

The same author elaborates further on the problem in his later work [18]. He correctly identifies two problems. The first problem is the assignment of all archived Web pages (*a capture*) to the same timepoint. The second problem is the need for comparison of two archived versions of the same Web page in order to assess the quality of the Web archive. To our knowledge, we are the first to address these issues [92, 33, 34]. Chapter 3 puts forward a Web archiving model which takes into account the two problems. The model distinguishes two cases. In the first case, the model maps a capture with all Web pages to an interval assuming that the users of the archive are interested with the same probability in every time point in the interval. Based on this assumption, the model suggests the blur metric as a stochastic estimate of the quality of the capture. In the second case, the model maps all invariant pages from the capture to the same timepoint. This is possible by introducing a second download for every page to check if the page changed or not. The quality in this case is measured by the coherence metrics. Saad et al.[82, 83, 11] propose an alternative model which strives for coherence with a single download per page. However, the authors assume the availability of the complete history of changes which is not realistic.

Given the importance of the changes for Web archives, we must know or at least predict when changes happen.

The easiest way would be to let Web site owners inform about changes. HTTP [40] and Sitemap [85] protocols provide mechanisms for that.

HTTP-based Web servers deliver Web pages as responses to GET or POST method requests. Web crawlers usually use the GET method requests to download pages (the POST method requests are common only for Deep Web crawlers). As a response, the Web server can either deliver the Web page or infrom the crawler that the page has not changed, i.e. send a HTTP **304 Not Modified** status. This is conditional GET mechanism that arises in two cases:

- **ETag/If-None-Match** transaction. At the first visit of a Web page, the Web server sends a ETag value, placed in the ETag header of the HTTP response. From then on, the Web crawler may send back the same ETag value in the If-None-Match header of the HTTP request. Listing 2.5 and Listing 2.6 are examples for the useage of the two header fields.

- **If-Modified-Since** header. The Web crawler sends a timestamp in the If-Modified-Since header of the HTTP request. The Web server delivers the Web page only the page has changed since the specified date. Listing 2.7 shows a an example value for the If-Modified-Since header.

```
ETag: "874897f4ca7c876b7e"
```

Listing 2.5: ETag Header of an HTTP Response

```
If-None-Match: "874897f4ca7c876b7e"
```

Listing 2.6: If-None-Match Header of an HTTP Request

```
If-Modifed-Since: Thu, 26 Apr 2012 21:03:29 CET
```

Listing 2.7: If-Modified-Since Header of an HTTP Request

The conditional GET mechanism is very useful for decreasing network traffic and reducing the load on the Web servers. We implement the conditional GET in our prototype. However, HTTP protocol does not help predicting changes. The protocol requires HTTP requests from Web crawlers in order to inform for changes.

In contrast to the HTTP protocol, sitemaps can improve change prediction. As mentioned before, the **changefreq** field contains the change frequencies of Web pages. The changefreq field can have one of the following values *always, hourly, daily, weekly, monthly, never*. However, not all Web sites have adopted the protocol. Even the existing sitemaps may not provide exact information. Still, sitemaps with correct change frequencies are very useful. They help improve the quality of Web archives. Therefore, our prototype supports the Sitemap protocol.

In any case, we need more reliable mechanisms for predicting changes. The only way how we can do that is to build a prediction model based on history of changes. We distinguish two directions: prediction of changes with a stochastic process and prediction of changes with change patterns. The choice depends on the time granularity of the history.

Change patterns are appropriate if it is possible to build a histogram of changes on an hourly basis [87, 11].

Sia et al. [87] show that patterns repeat daily due to the daily periodicity of human activity. They gather the posting history of 5166 blogs and group the posts of each blog in a histogram with 24 bins, each bin corresponding to an hour between 0 and 23. After applying K-means clustering algorithm, the authors identify 12 clusters of similar

histograms. The peaks of posting acitvity differ among clusters. Some clusters exhibit peak of activity during the morning, others during the afternoon.

In a similar fashion, Ben Saad and Grančarski collect change patters of Web pages. They too create aggregate the changes of each Web page in a histogram with 24 bins. Interestingly, they distinguish between work days, weekends, and holidays. Thus, three histograms correspond to the change patterns of each page. The Web pages are crawled from the Web site of the French National TV[12]. In order to gather enough data for the patterns, the authors select 100 Web pages that are crawled every hour during one month.

If a fine grained history is not readily at hand, the changes of a page p are typically modeled with a homogeneous Poisson process [14, 24]. The process has an *average change rate* parameter λ and the distribution of the number of changes of the page p per time unit Δ is

$$P[\text{number of changes of } p \text{ in } \Delta \text{ is } k] = \frac{e^{-\lambda\Delta}(\Delta\lambda)^k}{k!}.$$

The change rate λ is estimated from the observed changes of the page p. Intuitively, if X is the number of the observed changes and T the time interval the observations took place, then the estimated change rate $\hat{\lambda}$ is

$$\hat{\lambda} = \frac{X}{T}.$$

Cho and Garcia-Molina [24] show that this estimator reports smaller values for the change rate if the number of the observations is smaller than the number of changes. To tackle this issue, they propose two improved estimators based on the properties of the Poisson process. The first estimator assumes that the Web page is accessed n times in regular time interval Δ. If the properties of the Poisson process are applied directly, the resulting estimator $\frac{1}{\Delta}\log\left(\frac{n-X}{n}\right)$ has a mathematical singularity for $X = n$ (every access to the page detects a change). Therefore, the authors add a constant (0.5) in the definition of a smoothed form of the estimator. Then the estimator has the following form:

$$\hat{\lambda} = -\frac{1}{\Delta}\log\left(\frac{n-X+0.5}{n+0.5}\right).$$

The second estimator deals with irregular accesses to Web pages. It is not a closed-form expression, instead it is an algorithm that finds the value for λ with the maximum likelihood given the observations of both intervals with change and without change.

[12] http://www.francetv.fr/

Mathematically, the estimator finds out such λ that maximizes the probability

$$\prod_{i=0}^{X}(1-e^{-\lambda\Delta_i})\cdot\prod_{j=0}^{n-X}(e^{-\lambda\Delta_j}),$$

where Δ_i is the i-th interval with a change and Δ_j is the j-th interval without a change.

Independenly from Cho and Garcia-Molina, Matloff [60] reaches to similar solutions for the change rate estimator. The author as well suggests two estimators, but both of them for the case with regular access intervals. The first estimator is identical to the estimator proposed by Cho and Garcia-Molina (without the additional constant). As discussed before, the estimator has a mathematical singularity. The author as well proposes an alternative estimator without a closed-form expression which is a solution of an exponential equation.

In later studies [87, 89] non-homogeneous Poisson process was proposed as well. In this case, the change rate λ becomes a function of time $\lambda(t)$. Sia et al. [87] use histograms of daily changes to define the function $\lambda(t)$. Singh [89] suggests the Weibull distribution for the estimation of $\lambda(t)$ assuming that time interval of observations can be split into intervals where $\lambda(t)$ is monotonous:

$$\lambda(t)=\frac{\beta}{\eta}\left(\frac{t}{\eta}\right)^{\beta-1},$$

where β and η are parameters which can be estimated with the maximum likelihood estimator of the Weibull distribution.

In this thesis, we decide for a homogeneous Poisson process. It is a general model that describes well enough the change behaviour of any Web page. It achieves high accuracy with relatively little training data. In contrast, detecting periodic change patterns requires constant monitoring of all Web pages. At the same time, periodic change patterns focus on the daily distribution of changes. This information does not change the preferance for download positions closer to the middle of the crawl. The estimation of the number of changes per day is more important than the time distribution of changes during a day. This holds especially for crawls which span several days or even weeks. They rely on estimation of the change rates of the Web pages for the scheduling the downloads.

Since we do not always possess the history of all Web pages, we can not estimate the change rates for all pages. Then we can substitute the estimator with machine learning techniques. This is possible because of the correlations between various properties of the Web page and its change frequency. Douglis et al. [37] have been the first to observe

such correlations. Their list of features includes **page type**, **page size**, **top-level domain**, and **number of references**. Later Fetterly et al. [39] include **number of words** in list of possible features.

Tan et al. [95] use an extensive list of features which can be broken down into several categories:

- **Content features**: page type, page size, number of words, number of images, number of tables, and 10 features representing the textual content of the page;
- **URL features**: top-level domain, depth in the hierarchy, words in the URL;
- **Linkage features**: PageRank, number of inlinks, number of outlinks, number of email addresses;
- **Dynamic content features**: change of any of the following: the content, the number of images, the number of tables, and the page size;
- **Dynamic linkage features**: change of any of the following: PageRank, the number of the inlinks, and the number of the outlinks of the email addresses.

They apply the Repeated Bisection Clustering algorithm [50] to obtain hierarchical clusters of pages which change in a similar manner. The same authors later extend their study to [94] they extend the list of features with **ranking features** that describe users' behaviour taken from query logs.

Chen et al. [21] take a different direction. They identify association rules (positive and negative) based on the hierarchical structure of the Web site. They focus on the negative association rules, i.e. they look for Web pages that rarely change together.

In this section we reviewed various approaches for identifying changes in Web pages. We covered mechanisms for cooperation with the Web sites, prediction models based on Poisson processes and histograms as well as machine learning techniques with Web pages specific features. Most of them are used in incremental crawling strategies and aim at predicting changes at a specific timepoint. This helps the crawler decide whether to download a Web page or not. In contrast, we are interested in predicting changes in a time interval, so that a complete download schedule can be prepared. Web crawling strategies as part of the Web crawlers are discussed in detail in the next section.

2.3. Web Crawling

Web crawlers are computer programs that visit and download Web pages in an automatic manner. The crawler starts the process with a specified set of URLs or **seeds**. Figure 2.1 shows a typical architecture of an Web crawler. A Web crawler consist of a **downloader**, a **queue**, a **scheduler**, and a **storage**. The downloader or the **fetcher** retrieves Web pages from the Web most often with the HTTP protocol, although other protocols, like FTP for example, may be also used. After a download of a page, new URLs are placed in the queue of URLs. The scheduler rearranges the URLs in the queue based on the download strategy of the crawler and the next URLs are fed into the downloader.

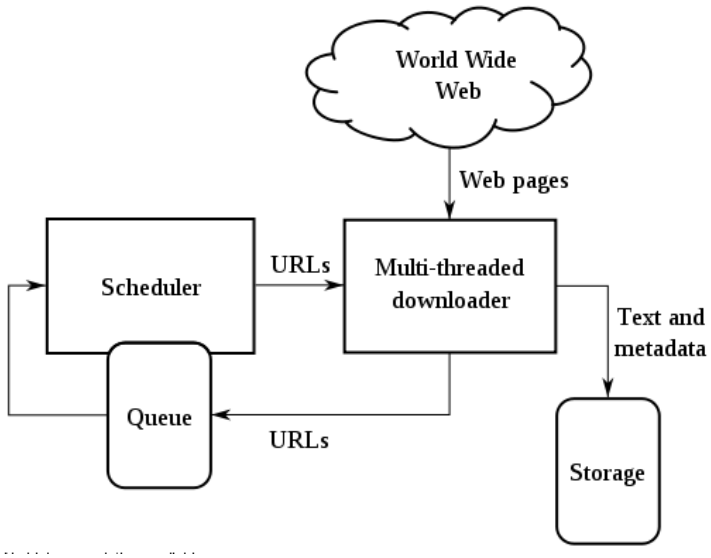


Figure 2.1.: Typical Web Crawler Architecture

This generic architecture of Web crawlers is the basis for Heritrix [65] — an extensible crawler used by various archiving institutions and national libraries. By default, Heritrix crawls sites in the breadth-first order of discovery. Heritrix is highly extensible in scope restriction, protocol based fetching, resource usage, and scheduling. Since our prototype is based on Heritrix, we give more detail on the Heritrix crawler in Chapter 7.

In the rest of the section, we focus on the various download strategies which Web crawlers employ. Olston and Najork[74] identify two types of crawls - **batch crawls** and **incremental crawls**.

Batch Crawls

Batch crawls are crawls with a single download per page, limited in time, and executed in regular intervals. Such crawls' aim is coverage. If pages have weights, the goal is highest weighted coverage. This is normally the case with search engines. The most common weight is PageRank [76].

Najork and Wiener [66] demonstrate that breadth-first strategy yields best results in regard with coverage. The strategy downoads the Web pages in the order of their discovery following the FIFO principle. Since pages with high PageRank have a greater chance for discovery, they are visited early in the crawl. This is crucial when the crawl is limited in time and resources.

Cho and Schonfeld [29] reformulate the problem of coverage by introducing the RankMass measure. The RankMass of a set of Web pages D is the sum of the PageRank of each page from D:

$$RankMass(D) = \sum_{p \in D} PageRank(p).$$

The coverage problem becomes an optimization problem: given the size N, find out the set D with the maximum RankMass. The authors suggest an algorithm that predicts the PageRank for each discovered URL and downloads the URL with the highest prediction. Similarly, the OPIC algorithm [1] predicts pages' popularity on-the-fly during crawling and chooses the page with the highest popularity.

Zheng et al. [100] improve the coverage of important pages by carefully selecting a set of seeds. They use a graph-based algorithm which chooses such Web pages for seeds, that have least distance to important Web pages.

Web archives, in constrast, consider every page equally important as long as the page belongs to a predefined scope. The scope can be defined by a top-level domain, a set of Web sites, or a topic. Batch crawls start with an initial set of seeds. Normally, the seeds are home pages of Web sites. One can either specify them manually or use links from Web directories, such as the Open Directory[13]. In case all relevant pages have equal weights, Masanès [59] distinguishes two cases: **extensive** and **intensive** crawling.

Extensive crawls aim at the top level pages from various Web sites. This is the case for **focused** and **topical crawling**. Both focused and topical crawlers download pages relevant to a user-defined set of topics. They classify discovered URLs during the crawl and decide whether to follow the link or not.

[13] http://dmoz.org

According to Menczer [56] the difference between the focused and the topical crawlers is in the training of the classifier. Focused crawlers train the classifier before the start of the crawl based on a large set of labeled examples for each topic. For example, Web directories provide a taxonomy of topics and corresponding examples. Topical crawlers, on the other hand, train their classfier during the crawl. They start with a limited set of labled examples and/or a short description of the topic, for example a keyword query.

Chakrabarti et al.[20] are the first to introduce focused crawling. Their crawler uses a text classifier pre-trained with labeled examples from any existing Web directory. The classifier is refined with a user-specified set of Web pages, for example bookmarks. This enables the crawler to suggest relevant topics to the user who either approves or discards the topics. Once the set of relevant topics is fixed, the crawler starts visiting Web pages and discovering new URLs. The decision whether the crawler visits a newly discovered Web page depends on the relevance of the parent Web page. In thit regard, the authors suggest two approaches: **hard rules** and **soft rules**. If hard rules are applied, the classifier chooses one topic for each visited Web page. If the topic is relevant, then the crawler follows all outgoing links in the page. If soft rules are applied, the crawler maintains a relevance score for each URL and stores it in a priority queue. Additionally, hub and authority scores are computed and pages with high hub scores are promoted in the queue. This ensures the download of Web pages that have links to authoritative sources.

An early forerunner of a topical crawler is proposed by Cho et al. [27]. The crawler does not support a classifier, instead it performs a simple check to determine if a newly discovered URL is relevant or not. The crawler follows a link in a visited Web page, only if the anchor text of the link contains a pre-defined keyword.

In contrast, the BINGO! system [90, 91] is a sophisticated solution for topical crawling. The system's input is a small hierarchy of categories with example URLs as seeds. User's bookmarks serve this purpose very well. The crawler has a set of binary classifiers (support vector machines) with one classifier per category. Since the initial training data is scarce, the BINGO! system periodically alternates between **learning phase** and **harvesting phase**. During the learning phase, the crawler retrains its classifiers with previously captured Web pages as training data. This phase also includes automatic discovery of topic-specific terms, which are afterwards used as features. During the harvesting phase, the crawler applies breadth-first strategy, starting from Web pages with high hub scores.

A different paradigm for topical crawling is InfoSpider [63, 64]. It consists of on a number of distributed agents which interact with the environment (follow links in the

Web). The agents are modeled like living organisms: they are born with initial amount of energy, learn, adapt, reproduce, and eventually die. Each interaction (following a link on the Web) has an energy cost and reward value (the relevance of the link to the topic). The agents exploit link topology and text similarity [62] to find the similar pages on-the-fly. Aggarwal et al. [4] proposed adaptive algorithms to improve the results of the topical crawlers.

Intensive crawls aim at all pages in one or several Web sites, for example deep Web crawls [67, 99] and crawls of top-level domains [9]. Deep Web crawls try to access information hidden in databases behind Web servers. They identify Web sites with graphical interfaces for queries [10], populate the fields and fire HTTP POST requests to the Web server. (For comparison, the extensive crawlers send GET requests). As access to full content is not always possible, Masanès [59] recommends cooperation with the Web site owners for supplementing the crawls.

Incremental Crawls

Incremental crawls are continuous. At every iteration, they decide either to a download a new page or to a revisit an old page. This depends on the purpose of the crawl. Olston and Najork identify two goals: maintaining fresh snapshots of the Web and capturing as many updates as possible. Search engines employ crawls which strive for **freshness**. Web archives aim at capturing as many updates as possible.

Freshness $F(p,t)$ of a page p can be defined either as a binary or as a continuous metric. The binary definition considers a page p at time point t fresh if its captured version is identical to its live version:

$$F(p,t) = \begin{cases} 1 & \text{if the captured version of } p \text{ is the same as the live version,} \\ 0 & \text{otherwise} \end{cases}$$

The freshness metrics extends to a set S of Web pages $p_0, \ldots, p_n$ as the average freshness of all pages:

$$F(S,t) = \frac{1}{n} \sum_{i=0}^{0} F(p_i,t).$$

Crawling strtegies for fresh snapshots typically maximize the freshness averaged over time $\bar{f}(S)$:

$$\bar{F}(S) = \lim_{t \to \infty} \frac{1}{t} \int_{0}^{t} F(S,t)t.$$

They optimize the freshness by assigning a revisit **frequency** f for each page p. Typically, it is assumed that a page changes according to a Poisson process with an average change rate λ [30, 14, 24, 23]. Cho and Garcia-Molina [25] investigate the relation between the revisit frequency and the change rate. They show that crawling strategies with uniform download frequency perform better in terms of freshness than crawling strategies where the revisit frequency of a page is proportional to the page's change rate.

An optimal strategy is proposed as well. This strategy has a counter-intuitive property: pages which change almost continuously are never revisited ($f = 0$). Such pages do not contribute to the overall freshness, since they become stale at the very next moment after their revisit. Therefore, it makes sense to revisit other pages that will remain fresh for longer periods of time.

In the same article, the authors introduce a continuous model for freshness, namely **age** $A(p,t)$. The age $A(p,t)$ is the amount of time since the time t_c of first change of p after its last download:

$$A(p,t) = \begin{cases} 0 & \text{if the captured version of } p \text{ is the same as the live version,} \\ t - t_c & \text{otherwise} \end{cases}$$

Analogically to the definition of the freshness metrics, the definition of the age extends to a set S of Web pages:

$$A(S,t) = \frac{1}{n} \sum_{i=0}^{0} A(p_i,t).$$

The crawling strategies with focus on the age, assign revisit the Web pages such that the time-averaged age is minimized. In this case, the revisits frequencies of the Web pages are proportional to their change rates.

Olston and Pandey [75] suggest a different strategy which takes into account long-lived and ephemeral content. Their model of content change consists of three types behaviour:

- **static behaviour**: the content stays the same,
- **churn behaviour**: the content is overwritten,
- **scroll behaviour**: new content is appended.

To characterize the change behaviour of the Web pages, the authors split Web pages into k-gram fragments on word level (*shingles* [16]). Shingles with long lifetime mark scroll behaviour, while short-lived shingles mark churn behaviour.

In the paper, revisits of pages with scroll behaviour are prefered to revisits of pages with churn behaviour. It is assumed that two versions of a page with scroll change behaviour will have higher Jaccard distance than two versions of a page with churn behaviour, given that a set of shingles $S(p)$ is a valid representation of the page p. The Jaccard distance $J_\delta(A,B)$ between two sets A and B is defined as:

$$J_\delta(A,B) = 1 - \frac{|A \cap B|}{|A \cup B|}.$$

The authors introduce the **divergence** $D(p)$ metrics which defined as the Jaccard distance between the live version of p and its last captured version. In a similar fashion to Cho and Garcia-Molina [25], Olston and Pandey propose a strategy which optimizes for divergence averaged over time.

Continuous crawls which maximize the number of updates are more suitable for Web archiving. The best strategy is to allocate revisits to pages proportionally to their change rate. Pandey et al. [78] add weights to pages and propose a crawling strategy that distributes m revisits to n pages ($m >> n$) aiming at least missed changes. However, this problem is NP-hard and an already existing approximation algorithm is applied for revisit allocation.

In a follow up work Pandey et al. [77] introduce the concept of **timeliness** which describes the acceptable delay in capturing changes: the higher the timeliness, the shorter the acceptable delay. This is expressed mathematically by the **urgency** function which is defined according to the user requirements. Three alternative definitions are proposed in the paper:

- **uniform urgency**: the urgency is constant, independent of the time (timeliness is low);
- **exponential decay**: the urgency decreases exponentially with the time (timeliness is high);
- **sliding window**: the urgency is 1 for a predefined interval, then drops to 0 (timeliness is high, but allows for a grace period).

According to this model, revisits of pages with high urgency are prioritized. The authors claim that an optimal algorithm is too expensive and suggest an approximation algorithm.

All of the described metrics and strategies have been implemented in the context of Web search engines. In the context of Web archiving, Sigurdsson [88] presents

the technical challenges and implementation issues on enabling Heritrix to support incremental crawling.

In the thesis, we combine features of both batch and incremental crawls. With respect to the number of downloads, our crawls are intensive batch crawls. Single-visit strategies download every page once, visit-revisit strategies download the pages twice, and completeness is a must. The blur and coherence metrics share similarities with binary and continuous freshness. Freshness, however, is defined for incremental crawls only. Batch crawls lack quality metrics which take into account the changes of the Web pages. The notaions blur and coherence introduced in this thesis address that problem.

2.4. Visualization of Changes in Web Archives

Visual clues of changes in Web archives provide better intuition for changes than mathematical models. We can split the available visualization tools into two groups: tools focusing on the changes inside the pages and tools focusing on the changes of the graph.

The first group includes Zoetrope [2], Vi-Diff [79], WebCQ [57], and a history browser [49]. All of them detect changes in the Web pages and highlight them during presentation. These tools analyze both the text and the DOM structure of a Web page. WebCQ provides a tabular summary of the changes of a page. Vi-Diff and the history browser make sure that the link navigation is meaningful. Zoetrope supports timeline visualization of the changes of selected Web pages.

The second group consists of WEEV [22], WebRelievo [96], and TimeSlices [48]. They display images of the whole graph at different time points. WEEV and WebRelievo put the images next to each other, while TimeSlices arranges the images in a 3D enabling the user to zoom in and zoom out. However, the huge number of displayed hyperlinks reduces the visibility and interpretability.

Lately, graphical interfaces to Web archives have been adopted in areas like spam detection and knowledge harvesting. Benczur et al. [12] enable archivists to denote suspicious Web pages from an Web archive as spam. Later, these Web pages are used as training data for spam detection [38]. Wang et al. [97] use Web archives to identify, disambiguate and visualize temporal facts. Their graphical interface focuses on the entities and the relations found in the archive.

Chapter 3.

Web Archiving Model

3.1. Introduction

Web archives, very much like traditional archives and libraries, provide excellent material for both researchers and professionals. On one hand, sociologists, politologists, media analysts may be able to detect global trends in the development of the information society. On the other hand, experts on intellectual property and compliance may be able to prove or disprove claims based on archived Web sites. The potential of Web archives is already recognized: many national libraries es (e.g., `www.loc.gov`, `www.webarchive.org.uk`, `netarkivet.dk`, `www.bnf.fr`, `www.webarchiv.cz`, etc) and organizations like the Internet Archive (`archive.org`) and the European Archive (`europarchive.org`) take active roles in preserving the Web. However, the changing nature of the Web poses new challenges to the archiving institutions. We may be tempted to assume that a Web archive is an exact image of a Web site as of some time point in the past. In practice, however, archiving an entire Web site takes time during which many pages change. The changes affect the mutual consistency of the pages and makes interpreting the archive difficult. A real-life example is a lawsuit about intellectual property rights [80]. The judge disproved a Web archive as evidence in court due to the lack of quality. A traditional archive would have been approved as evidence. To close the gap between Web archives and traditional archives, we must maintain Web archives with a high standard of data quality.

While the issue of Web-archive quality is obvious, it is unclear how to formalize the problem and address it technically. This chapter presents a model which enables quality assurance of Web archives. The model includes several properties of the archive and the archiving process: the archive's organization, the access to the archive, and the types

of crawling strategies. Together, these properties allow us to define quality metrics and improve the crawling strategies accordingly. We organize the model, the quality metrics, and the crawling strategies in the SHARC framework for data quality in Web archiving.

3.2. Concepts

Organization of Web Archives

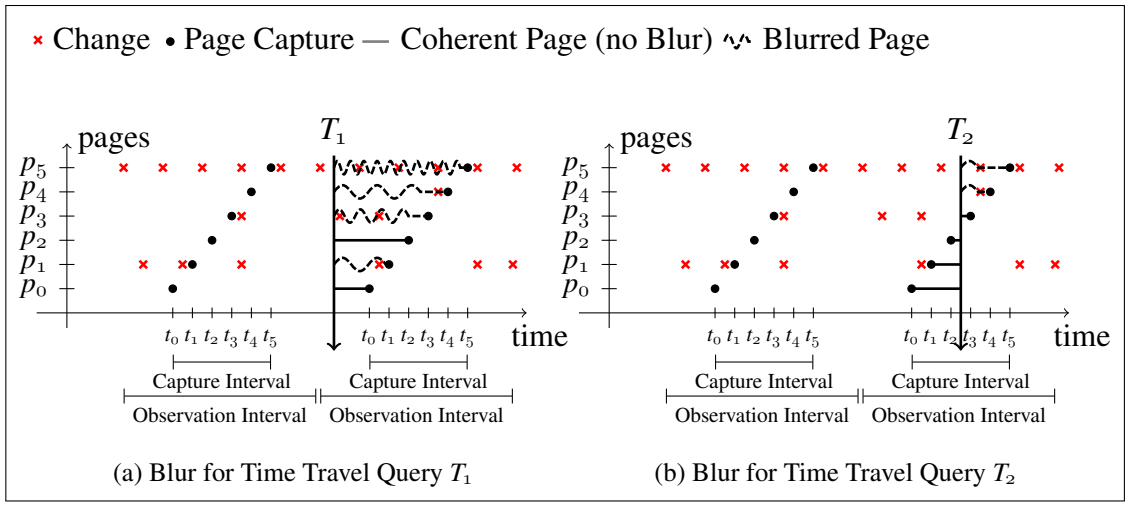
Web archives consist of series of *captures*. These are periodic versions of a Web site. A *crawler* produces a capture by downloading all pages of the site. Every crawler has a *crawling strategy* which determines the number and the order of the downloads. The interval between the first and the last download is called *capture interval*. The captures in Web archives correspond to a time interval, in contrast to the records in traditional archives which correspond to a time point. Although, we would like to have snapshots of the Web site for every time point, this is practically impossible. To tackle this issue, we propose two alternatives approaches for organization of the Web archive. In the first approach, the Web archive maximizes the number of captures, but gives only stochastic guarantees about its quality. In the second approach, the Web archive gives deterministic guarantees for quality by reducing in half the number of captures.

If the number of captures is more important than the quality guarantees, we use *single-visit* crawling strategies. They are the fastest possible strategies which at the same time ensure completeness of the captures. These strategies are suitable for archiving news Web sites which change often and need many captures to cover all changes. In contrast, a Web archive of Web sites of governmental institutions requires credibility. Quality guarantees are crucial. In this case, we use *visit-revisit* crawling strategies. They double the time of the crawl but precisely detect all changed Web pages.

In both approaches, the crawler must observe the *politeness requirements* of a site, with pauses of several seconds or even a minute between successive HTTP requests. Thus, an entire site capture may span several days. (The crawler may crawl many sites in parallel for high throughput.) When a new site crawl starts, we may assume that the URLs of all pages are known upfront. This assumption is realistic if a previous crawl exists or the Web site suppports the Sitemap protocol [85]. If none of the above is available, then, we assume that at least one entry page is known from which the crawl can explore the site's page graph. We may assume that the total number of pages in a site can be estimated at the start of the crawl, based on site properties such as domain name or attributes obtained

by the HTTP reply when fetching the site's entry page.

Note that pages may change during a crawl. The longer the crawl duration, the more likely it is that a non-negligible fraction of the site's pages change once or several times. Figure 3.1 illustrates this situation for single-visit crawls and Figure 3.2 illustrates the situation for visit-revisit crawls. The time points when a page is downloaded are marked by a bullet, the time points when a page changes are marked by a cross. Note that a capture with a visit-revisit crawl needs almost doubles the downloads. Both figures show two crawls, each yielding a separate capture that will be stored in the Web archive for later retrieval by analysts and other users.



(a) Blur for Time Travel Query T_1 (b) Blur for Time Travel Query T_2

Figure 3.1.: Coherent vs. Blurred Pages with Single-Visit Crawling

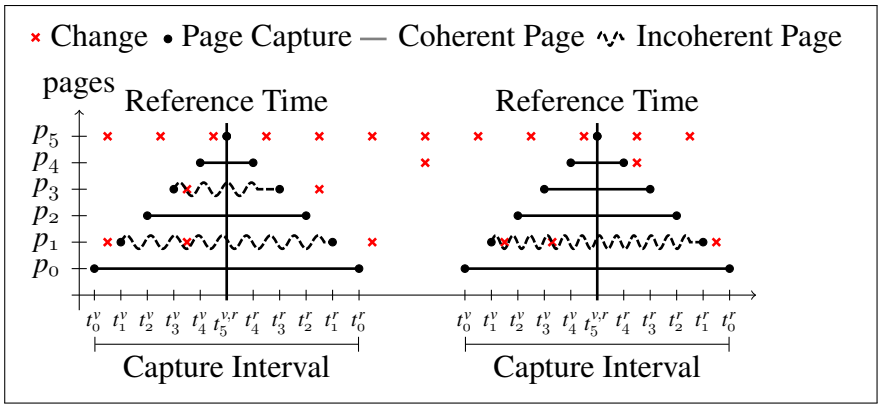


Figure 3.2.: Coherent vs. Incoherent Pages with Visit-Revisit Crawling

Access to Web Archives

In these settings, a Web archive is accessed by *time-travel queries*, asking for the site *as of* a given time point of interest to the user. Figure 3.1 shows two such requests, denoted by vertical arrows, for time points T_1 and T_2 which fall before the capture (Figure 3.1a) and within the interval between page-capturing time points t_2 and t_3 (Figure 3.1b). The archive may not have versions of the pages as of the exact requested time. The user's request for time T is then mapped either to the most recent available capture whose timestamp does not exceed T or to the nearest capture in the past or future (whichever is closer to T). This mapping defines for each capture an *observation interval*: the capture is returned for all time-travel queries that fall into the same observation interval. Figure 3.1 shows a possible choice for the observation intervals of the two captures. Observation intervals based on the most recent available capture correspond to the simplest standard semantics in temporal database systems [86]. Observation intervals based on the closest capture (preceding or following the user's time target) may appear non-standard, but make sense in our Web archive setting because the user's time point of interest may often be fuzzy or a crude estimate for exploration purposes. For example, when a sociologist wants to investigate opinions of a social group on a particular topic using the content of a site as of May 2001 (which could technically be interpreted as mid May, i.e., May 15, if a real time point is needed), she may be equally happy with a capture from April 28 or June 3 if the site was not captured during May.

Crawling Strategies

For individual time-travel queries some of the pages are "blurred" and some are "incoherent". By *blur* we refer to the magnitude of change in the time span between the query timestamp and the time of the actual page capture. The farther the two points are apart and the more the page changes (on average), the more blurred the page would appear to the user. For example, if the time-travel request is for May 15, a page that was captured on May 10 would appear less blurred than a page that was captured on May 1 or May 31. We will formalize this measure in Chapter 4. By *incoherence* we refer to the entire site's page changes between the query timestamp and the page capture. If some page has changed between these two time points, it may appear incoherent with respect to other pages. Conversely, if we manage to capture a set of potentially interrelated pages such that there is no change at all between their capturing points and the timestamp of user

request, then this set would be perfectly coherent. The pages jointly appear as if they were instantaneously captured in the very same state of the Web site. Pages without any changes in the critical time span are coherent and indicated by solid lines (Figure 3.1 and Figure 3.2). We will formalize the notion of coherence in Chapter 5.

In the SHARC framework, we want to either minimize the blur or maximize the coherence of captures. Each of these two goals requires additional information.

To minimize the blur, we need the timestamps of the time-travel queries of all archive users. As we can not know the timestamps at the time of the crawl, we assume that they are uniformly distributed in the observation interval. Hence, we will aim for a stochastic notion of archive quality.

To maximize the coherence, we need to know exactly which pages changed. Thus, we visit each page twice, where the second visit serves to check for changes. This *visit-revisit* approach is illustrated in Figure 3.2. The figure shows a strategy where all the revisits of pages follow all visits. Even if the HTTP protocol information does not reliably indicate whether a page is modified or not, we can now easily compare two versions of the same page and test for invariance (perhaps ignoring insignificant changes such as banner ads or auto-generated footers). If for a given set of pages (ideally the entire site), none shows any changes between visit and revisit, then this entire set is coherent - as if it were instantaneously captured in the middle of the crawl – denoted as “reference time” in the figure. In the thesis, we will consider only strategies with this two-phase structure: all revisits following all visits. Note, however, that the order in which individual pages are visited in each phase is an important degree of freedom in optimizing the crawl schedule.

Single-visit strategies have lower crawl cost than visit-revisit strategies and are good enough to minimize the stochastic notion of blur. However, an archive user cannot be sure if an individual page is coherent or not (i.e., did not or did change between query timestamp and page capture). This uncertainty is not a problem for exploratory usage of the archive. However, in use cases where we need to be sure that an archived snapshot reflects a Web site as of a specific time in the past (e.g., for legal purposes), we need to employ the visit-revisit method.

3.3. Model of Changes

In devising suitable strategies for scheduling the visits of a site’s pages, we need to have some information about how often, and perhaps even when, a page typically changes.

To this end, we employ statistical prediction models. Following the state of the art [24, 26, 89, 51], we assume that pages undergo changes according to a *Poisson process* with *change rate* λ.

The number of changes of the page p per time unit Δ is distributed according to a Poisson distribution with parameter λ:

$$P[\text{number of changes of } p \text{ in } \Delta \text{ is } k] = \frac{e^{-\lambda\Delta}(\Delta\lambda)^k}{k!}.$$

It is equivalent to postulate that the time between two successive changes of page p is exponentially distributed with parameter λ:

$$P[\text{time between changes of } p \text{ is less than } \Delta] = 1 - e^{-\lambda\Delta}.$$

We can then train a classifier or regression model to predict the specific rate of a page, based on features of the page: its MIME type, depth in the site graph relative to the entry point, URL string, and so on. The classifier allows us to predict the probability or "risk" that there will be a change of a page in the time span between capturing it and a time-travel access, and also the expected number of changes in that interval or a quantile for the number of changes. Chapter 6 is dedicated to our change prediction model.

3.4. Datasets

In the next chapters we describe the single-visit and the visit-revisit strategies on the Web archiving model. Since in the both cases we evaluate the strategies over the same real-world datasets we introduce the datasets in this chapter.

The real-world datasets consist of repeated crawls of different-sized domains including `mpi-inf.mpg.de` (MPII), `dmoz.org` (DMOZ), and sites from the .uk.gov collection (UKGOV). They are summarized in Table 3.1. The datasets span long periods (a year or longer) and consist of captures (weekly or even daily) of entire sites. The MPII dataset is a Web archive of an academic Web site (Max Planck Institute for Informatics), the UKGOV dataset consists of a number of governmental Web sites (UK government), the DMOZ dataset is a Web directory.

The MPII dataset constitutes crawls of our Web server. The "home" Web server allowed us to crawl it frequently and aggressively (without respecting the politeness delays). We crawled it on daily basis for one year. The DMOZ dataset represents a large Web site with 7 subsites (topic categories) that change frequently and subsites that change infrequently.

Every day we crawled one of the subsites. In a week's time we got a complete capture with all 7 subsites. The UKGOV dataset spans two years and 7 months and is, to the best of our knowledge, the most comprehensive, freely available Web archive reference collection.

Dataset	Abbreviation	Web Site	Periodicity	Time Range	Pages	Changing Pages
MPII	MPII	mpi-inf.mpg.de	daily	08.09–07.10	72,071	1,356
DMOZ	DMOZ	dmoz.org	weekly	10.09–07.10	177,446	50,855
UKGOV	MOD	mod.uk	weekly	08.03–02.06	10,047	5,988
UKGOV	DFID	dfid.gov.uk	weekly	08.03–02.06	2,186	1,131
UKGOV	ARMY	army.mod.uk	weekly	08.03–02.06	37,330	15,259
UKGOV	RAF	raf.mod.uk	weekly	08.03–02.06	27,836	4,286
UKGOV	DH	dh.gov.uk	weekly	08.03–02.06	15,884	12,203

Table 3.1.: Datasets

3.5. Summary

In this chapter we have presented a Web archiving model which takes into account the archive's organization, the access to the archive, and the types of crawling strategies. We have introduced concepts which describe the aforementioned properties. This allowed us to reason about the quality of the archive and to define the blur and the coherence metrics for measuring the quality. In Chapter 4 and Chapter 5 we formalize the definitions and design strategies optimizing these metrics.

Chapter 4.

Single Visit Crawling Strategies

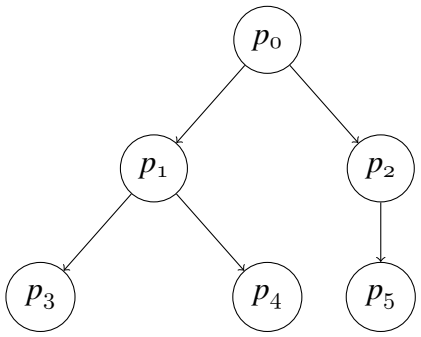
4.1. Introduction

In this chapter, we establish the stochastic metric blur for a given sites capture and develop the SHARC-Offline and SHARC-Online crawling strategies. Both strategies produce download schedules which aim at reducing the blur. SHARC-Offline assumes that all URLs of the pages in the Web site are known in advance. Thus, the strategy is able to prepares a complete download schedule before the start of the crawl. However, this is not a feasible solution in realistic run-time settings. Rather, the strategy a useful baseline for developing practically viable algorithms and assessing their quality. In contrast to SHARC-Offline, SHARC-Online is a crawling strategy that can be applied in real-life settings. SHARC-Online starts with a limited number of seeds (entry points to the Web site). During the crawl, the strategy discovers the URLs of other pages in the Web site. SHARC-Online updates its download schedule after every download, such that the schedule resembles SHARC-Offline. Still, this entails increase of the blur of the capture. Two factors influence the increase of the blur: the crawl duration and the distribution of the change rates among the Web pages. We investigate analyitically both factors. At the end of the chapter, we show experimentally that SHARC-Offline and SHARC-Online indeed increase the quality of the captures in comparison to the standard crawling strategies for Web archiving. First, though, we introduce the notation used throughout the chapter as well as an example Web site for illustrating the concepts and the algorithms we developed.

We assume that the Web archive consists of Web pages $p_0, \ldots, p_n$, which change according to the Poisson distribution with change rates $\lambda_0, \ldots, \lambda_n$ (the mean number of changes in a time unit). For ease of presentation, we assume that the identifiers (subscripts)

Page	Change Rate λ
p_0	0
p_1	1
p_2	2
p_3	3
p_4	4
p_5	5

(a) Change Rates



(b) Web Graph

Figure 4.1.: Example of a Web Site with Change Rates

of the pages are chosen so that $\lambda_0 \leq \cdots \leq \lambda_n$. We denote the least frequently changing page p_0 as the *coldest page*, and the most frequently changing page p_n as the *hottest page*. We assume that the download timepoints of the pages are equidistant with politeness delay Δ in between the downloads (the most typical scenario). To simplify mathematical expressions, we assume that the crawl starts at time 0 and the observation interval coincides with the capture interval: $[o_s, o_e] = [c_s, c_e] = [0, n\Delta]$. Later, in Section 4.3.3, we generalize the equations and omit the assumption.

Figure 4.1 presents an example that we use throughout the chapter. Change rate $\lambda = 0$ means virtually no changes for p_0. Strictly mathematically this is not possible, since Poisson model requires $\lambda > 0$. Here we assume a negligible small change rate $0 < \varepsilon \ll 1$.

4.2. Blur

The blur of a page and the blur of an entire site capture are key measures for assessing the quality of a single-visit Web archive.

Definition 4.1 (BLUR). *Let p_i be a Web page captured at time t_i. The* blur of the page *is the expected number of changes between t_i and query time t, averaged through observation interval $[0, n\Delta]$:*

$$B(p_i, t_i, n, \Delta) = \frac{1}{n\Delta} \int_0^{n\Delta} \lambda_i \cdot |t - t_i| dt = \frac{\lambda_i \omega(t_i, n, \Delta)}{n\Delta}, \tag{4.1}$$

where

$$\omega(t_i, n, \Delta) = t_i^2 - t_i n\Delta + \frac{(n\Delta)^2}{2}. \tag{4.2}$$

is the download schedule penalty.

Let $P = (p_0, \ldots, p_n)$ be Web pages captured at times $T = (t_0, t_1, \ldots, t_n)$. The blur of the archived capture *is the sum of the blur values of the individual pages:*

$$B(P,T,n,\Delta) = \frac{1}{n\Delta} \sum_{i=0}^{n} \lambda_i \omega(t_i,n,\Delta). \tag{4.3}$$

The blur indicates how many expected changes the explorer of the archive sees if she visits all the pages in the archive. We define the average blur as the average number of the expected changes explorer sees per page:

$$\bar{B}(P,T,n,\Delta) = \frac{1}{n(n+1)\Delta} \sum_{i=0}^{n} \lambda_i \omega(t_i,n,\Delta). \tag{4.4}$$

The blur of a Web page in the capture is the product of its change rate and $\omega(t_i,n,\Delta)$. $\omega(t_i,n,\Delta)$ depends on the download time and the length of the capture interval $n\Delta$ and does not depend on the page. Therefore $\omega(t_i,n,\Delta)$ can be interpreted as the penalty of downloading page p_i at time t_i.

Example 4.1 (Blur) *Consider the Web site in Figure 4.1 with download time t_i for page p_i (for example page p_3 is downloaded at time $t_3 = 3$). The blur of p_1 is*

$$B(p_1,1,5,1) = \frac{1 \cdot (1^2 - 1 \cdot 5 \cdot 1 + (5 \cdot 1)^2/2)}{5 \cdot 1} = 1.7.$$

Similarly $B(p_0,0,5,1) = 0$, $B(p_2,2,5,1) = 2.6$, $B(p_3,3,5,1) = 3.9$, $B(p_4,4,5,1) = 6.8$, $B(p_5,5,5,1) = 12.5$. The blur of the archive is

$$B(P,T,5,1) = 0 + 1.7 + 2.6 + 3.9 + 6.8 + 12.5 = 27.5$$

The average blur per page is $\bar{B}(P,T,5,1) \approx 4.58$.

Properties of the download schedule penalty allow us to reason about the lower and the upper bounds of the blur of a capture and the impact of the length of the delay intervals.

Theorem 4.1 (LOWER AND UPPER BOUNDS) *The lower bound of the blur is $\frac{n\Delta}{4} \sum_{i=0}^{n} \lambda_i$ and the upper bound is $\frac{n\Delta}{2} \sum_{i=0}^{n} \lambda_i$.*

$$\frac{n\Delta}{4} \sum_{i=0}^{n} \lambda_i < B(P,T,n,\Delta) < \frac{n\Delta}{2} \sum_{i=0}^{n} \lambda_i. \tag{4.5}$$

Proof of Theorem 4.1 *The proof of the bounds follows from substituting the schedule penalty in the blur formula with its minimum and its maximum. Let us recall that the schedule penalty is a quadratic function with minimum equal to* $\frac{(n\Delta)^2}{4}$ *and maximum equal to* $\frac{(n\Delta)^2}{2}$ *in the interval* $[0, n\Delta]$. □

One can interpret the lower bound of the blur as the blur of a fictitious crawl with coherent snapshots at every download time point. The blur is not 0 since the time-travel queries may happen between two download points and changes still may occur. The interpretation of the upper bound is similar. The upper bound is the blur of a fictitious crawl which at each download time point has snapshots as "blurred" as possible.

Theorem 4.2 (PROPERTIES OF THE SCHEDULE PENALTY) *The blur is proportional to download delay* Δ*, i.e.,*

$$B(P,T,n,\Delta) = \Delta B(P,T,n,1). \tag{4.6}$$

Proof of Theorem 4.2

The proof follows from the definitions of schedule penalty, blur, and quadratic function of penalty. □

4.3. SHARC-Offline Strategy

4.3.1. Optimal Download Schedule

Different download schedules result in different values of blur. We now investigate the optimal download schedule for archiving. Mathematically, for the given Web site $p_0, \ldots, p_n$ we will identify the optimal schedule $t_0, \ldots, t_n$, (a permutation of $0, \Delta, \ldots, n\Delta$) that minimizes the blur of the archive (cf. Equation (4.3)). In particular, we show that the pages that change most should be downloaded in the middle of the crawl.

Example 4.2 (OPTIMAL DOWNLOAD SCHEDULE.) *Consider again Example 4.1. The optimal download schedule is* $t_0 = 0$ *and* $t_1 = 5$ *(the outermost points of the interval) for the coldest (least changing) pages* p_0 *and* p_1*,* $t_2 = 1$ *and* $t_3 = 4$ *(the next outermost points) for the second and third least changing pages* p_2 *and* p_3*, followed by* $t_4 = 2$ *and* $t_5 = 3$ *for the hottest pages* p_4 *and* p_5*. The blur of the capture with optimal download schedule is* $B(P,T',5,1) = 22.7$.

Figure 4.2 illustrates the optimal download schedule where the change rate of the scheduled download is visualized as a line of length proportional to the change rate. The

visualization resembles an organ-pipes arrangement *with the highest pipes allocated in middle.*

Theorem 4.3 (OPTIMAL DOWNLOAD SCHEDULE) *Let* p_0, $p_1, \ldots, p_n$ *be the Web site such that* $\lambda_0 \leq \lambda_1 \leq \cdots \leq \lambda_n$. *Then the optimal download schedule* $t_0, \ldots, t_n$ *is defined by the following equation:*

$$t_i = \begin{cases} \frac{i}{2} & \text{if } i \text{ is even,} \\ n - \frac{i-1}{2} & \text{otherwise.} \end{cases} \tag{4.7}$$

for $i = 0, 1, \ldots, n$.

Proof of Theorem 4.3

The proof of Theorem 4.3 is based on three observations:

(i) λ_i *are ordered increasingly:* $\lambda_i \leq \lambda_{i+1}$,

(ii) Equation (4.7) orders $\omega(t_i, n, \Delta)$ *decreasingly:* $\omega(t_i, n, \Delta) \geq \omega(t_i, n, \Delta)$,

(iii) Equation (4.3) is minimized when λ_i *are ordered decreasingly and* $\omega(t_i, n, \Delta)$ *are ordered increasingly.*

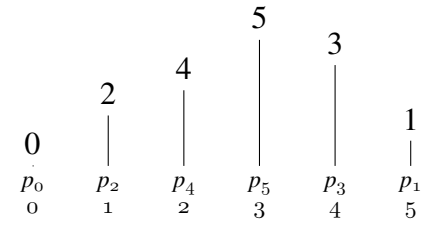


Figure 4.2.: Organ-Pipes Arrangement

λ_i *are scheduled increasingly because of the assumption of the theorem, and therefore case (i) is true.*

Function $\omega(t, n, \Delta)$ *is quadratic in t, with its minimum at* $t_{min} = (n\Delta)/2$. *Equation (4.7) schedules* t_i*s in such a way that* $\omega(t_n, n, \Delta)$ *is smallest. The greater the index i, the closer to the middle* t_i *is allocated. Therefore* $\omega(t_i, n, \Delta)$ *are scheduled decreasingly and Case (ii) is true.*

The proof of Case (iii) is given in Lemma 4.1 below. □

Lemma 4.1 *Let* $\lambda_0 \leq \lambda_1 \leq \cdots \leq \lambda_n$ *and* $\omega(t_0, n, \Delta) \geq \omega(t_1, n, \Delta) \geq \cdots \geq \omega(t_n, n, \Delta)$. *Let* $j_0, \ldots, j_n$ *be a permutation of* $0, \ldots, n$. *Then*

$$\sum_{m=0}^{n} \lambda_m \omega(t_m, n, \Delta) \leq \sum_{m=0}^{n} \lambda_{i_m} \omega(t_{j_m}, n, \Delta). \tag{4.8}$$

Proof of Lemma 4.1

Indeed, let $i_0,\ldots,i_n$ be the optimal permutation of $0,\ldots,n$ such that $(\lambda_{i_0},\ldots,\lambda_{i_n})$ and $(\omega(t_{j_0},n,\Delta),\ldots,\omega(t_{j_n},n,\Delta))$ minimize the right hand side of Equation (4.8).

The largest element of λs must be multiplied by the smallest element of $\omega(t_{i_l})$, otherwise the solution is not optimal. We prove this step by contradiction. Let

$$\begin{array}{cccccccc} (\lambda_{i_0}, & \ldots, & \lambda_{i_k}, & \ldots, & \lambda_{i_l}, & \ldots, & \lambda_{i_n}) \\ (\omega(t_{j_0},n,\Delta), & \ldots, & \omega(t_{j_k},n,\Delta), & \ldots, & \omega(t_{j_l},n,\Delta), & \ldots, & \omega(t_{j_n},n,\Delta)) \end{array}$$

be the optimal schedule, however $\lambda_{i_k}=\lambda_0$ (the smallest among all λ_is) and $\omega(t_{j_l},n,\Delta)=\omega(t_0,n,\Delta)$ (the largest among all $\omega(t_i,n,\Delta)$s). Then we can show that the by swapping λ_{i_k} with λ_{i_l} (or alternatively $\omega(t_{i_k},n,\Delta)$ with $\omega(t_{i_l},n,\Delta)$) we can further decrease the sum in Equation (4.8). Indeed, the sum without the swap is:

$$\sum_{\substack{m=1,\ldots,n \\ m\neq l, m\neq k}} \lambda_{i_m}\omega(t_{j_m},n,\Delta)+\lambda_0\omega(t_{j_k},n,\Delta)+\lambda_{i_l}\omega(t_0,n,\Delta) \tag{4.9}$$

The sum with the swap is:

$$\sum_{\substack{m=0,\ldots,n \\ m\neq l, m\neq k}} \lambda_{i_m}\omega(t_{j_m},n,\Delta)+\lambda_0\omega(t_0,n,\Delta)+\lambda_{i_k}\omega(t_{j_l},n,\Delta) \tag{4.10}$$

Since the first sums in Equations (4.9) and (4.10) are the same we reach the contradiction if we prove that

$$\lambda_0\omega(t_{j_k},n,\Delta) \quad + \quad \lambda_{i_l}\omega(t_0,n,\Delta) \quad > \quad \lambda_0\omega(t_0,n,\Delta) \quad + \quad \lambda_{i_k}\omega(t_{j_l},n,\Delta). \tag{4.11}$$

The left hand side (LHS) of Equation (4.11) is:

$$\begin{aligned} LHS &= \lambda_0\omega(t_{j_k},n,\Delta)+\lambda_{i_l}\omega(t_0,n,\Delta) \\ &= \lambda_0\Big(\omega(t_0,n,\Delta)+\big(\omega(t_{j_k},n,\Delta)-\omega(t_0,n,\Delta)\big)\Big)+\lambda_{i_l}\omega(t_0,n,\Delta) \\ &= \lambda_0\omega(t_0,n,\Delta)+\big(\lambda_{i_l}+(\lambda_0-\lambda_{i_l})\big)\times\big(\omega(t_{j_k},n,\Delta)-\omega(t_0,n,\Delta)\big)+\lambda_{i_l}\omega(t_0,n,\Delta) \\ &= \lambda_0\omega(t_0,n,\Delta)+\lambda_{i_l}\omega(t_{j_k},n,\Delta)+\big(\lambda_{i_l}-\lambda_0\big)\big(\omega(t_0,n,\Delta)-\omega(t_{j_k},n,\Delta)\big) \\ &= RHS+\textit{strictly positive number}, \end{aligned}$$

since λ_0 is the smallest among λ_is and $\omega(t_0,n,\Delta)$ is the largest among $\omega(t_j,n,\Delta)$s.

The proof of lemma follows with the help of mathematical induction. The induction basis is trivial. The optimal solution for n reduces to the optimal solution for $n-1$ elements, since the λ_0 must be multiplied with $\omega(t_0)$ in the optimal solution. □

4.3.2. SHARC-Offline Algorithm

Algorithm 1 depicts the algorithm of SHARC-Offline. Since all the pages are known and sorted in advance we need to scan all the pages only once to schedule the downloads. The pages with even indexes are scheduled increasingly according to their change rates on the left hand side of the organ-pipe, while the pages with odd indexes are scheduled decreasingly on the right hand side of the organ-pipe.

Algorithm 1 SHARC-Offline

Require: sorted pages $p_0, \ldots, p_n$
Ensure: download schedule $p_0^D, \ldots, p_n^D$
  for $i = 0$ to n **do**
    if i is even **then**
      $p_i^D = p_{i/2}$
    else
      $p_i^D = p_{n-(i-1)/2}$
    end if
  end for

4.3.3. General Observation Interval

In this section we generalize the notion of blur and the optimal download schedule for the case when the observation interval $[o_s, o_e]$ does not coincide with the capture interval. Then the *blur of a page* is

$$B(p_i, t_i, n, \Delta) = \frac{1}{o_e - o_s} \int_{o_s}^{o_e} \lambda_i \cdot |t - t_i| dt = \frac{\lambda_i \omega(t_i, o_s, o_e)}{o_e - o_s},$$

where

$$\omega(t_i, o_s, o_e) = t_i^2 - t_i(o_s + o_e) + \frac{o_s^2 + o_e^2}{2}$$

is the *generalized download schedule penalty* and the *blur of the archived capture* is the sum of the blur values of the individual pages (cf. Equation (4.3)).

Theorem 4.3 schedules the hottest pages in the middle of the capture interval (point $n\Delta/2$). In case the observation interval does not coincide with the capture interval and there are no restrictions for the start of the capture interval we should schedule the most changing pages around the middle of the observation interval (point $(o_s + o_e)/2$). We formalize it in the following theorem.

Theorem 4.4 *Let $t_0, \ldots, t_n$ be the optimal download schedule for Web site with $[0, n\Delta]$ observation interval. Then*

$$t_i + \frac{o_e + o_s - n\Delta}{2}$$

is the optimal download position for page p_i with $[o_s, o_e]$ observation interval.

When the observation interval and the capture intervals are fixed, the hottest pages should be allocated as close as possible to the middle point of the observation interval.

4.4. SHARC-Online Strategy

The SHARC-Offline strategy assumes that all URLs of the Web site are known in advance. In this section, we relax the assumption and develop SHARC-Online, an archive crawl optimization strategy with no or limited knowledge of the Web site. Starting with a given set of seeds SHARC-Online *incrementally* extracts the URLs of other pages from the downloaded pages and schedules the pages for download so the blur is minimal. We organize this section as follows. First, we explain the incremental detection of the Web site structure and discuss most common crawl strategies in Section 4.4.1. We develop the SHARC-Online strategy by example in Section 4.4.2. Finally, we formally define the SHARC-Online strategy and present the algorithm of the strategy in Sections 4.4.3 and 4.4.4.

4.4.1. Discovery of the Web Graph

Typically, crawlers do not know the URLs of the pages in a crawled site. Archive crawlers start with the download of a given set of URLs (seeds of the crawl), extract the URLs of the downloaded pages, and continue the process until all the documents are downloaded and no new URLs are detected. At any iteration the crawler keeps Downloaded-Detected lists (DD-lists) of URLs. The downloaded list of URLs consists of all URLs that are already crawled, while the detected list comprises the extracted from the downloaded pages but not yet downloaded URLs. Different crawl strategies schedule the URLs in a different manner. Below, we demonstrate the most popular crawl strategies: *Depth-First (DFS)* and *Breadth-First (BFS)* on the example Web graph in Figure 4.1.

Table 4.1a depicts the detection and downloads of Web pages of the Depth-First strategy. The strategy starts with the seed page p_0 and inserts it into the detected part of the DD-list (cf. p_0 in the iteration $I = 0$ in Table 4.1a). Then, it downloads the page (p_0 is moved

to the downloaded part of the DD-list, cf. $I = 1$ in the table), parses the HTML page, and inserts detected URLs p_1, p_2 into the detected part of the DD-lists. The Depth-First strategy inserts newly detected pages at the beginning of the detected list, thus the newly detected pages have higher priority for download (cf. iteration $I = 2$ in the table). In contrast, Breadth-First strategy appends newly discovered pages, assigning a higher priority for early detected pages (cf. Table 4.1).

I	DD-list	
	Downloaded	|Detected
0		$\|p_0$
1	p_0	$\|p_1, p_2$
2	p_0, p_1	$\|p_3, p_4, p_2$
3	p_0, p_1, p_3	$\|p_4, p_2$
4	p_0, p_1, p_3, p_4	$\|p_2$
5	p_0, p_1, p_3, p_4, p_2	$\|p_5$
6	$p_0, p_1, p_3, p_4, p_2, p_5$	|

(a) Depth-First Crawl Strategy (DFS)

I	DD-list	
	Downloaded	|Detected
0		$\|p_0$
1	p_0	$\|p_1, p_2$
2	p_0, p_1	$\|p_2, p_3, p_4$
3	p_0, p_1, p_2	$\|p_3, p_4, p_5$
4	p_0, p_1, p_2, p_3	$\|p_4, p_5$
5	p_0, p_1, p_2, p_3, p_4	$\|p_5$
6	$p_0, p_1, p_2, p_3, p_4, p_5$	|

(b) Breadth-First Crawl Strategy (BFS)

Table 4.1.: Popular Crawl Strategies

4.4.2. SHARC-Online Strategy by Example

At any given iteration, the crawler does not know all pages but only the pages of the Web site in the DD-list. Our SHARC-Online strategy optimizes the download and detection of the Web pages incrementally. Given the (estimated) size of the Web site, the SHARC-Online produces a download schedule that resembles the organ-pipe order of the SHARC-Offline strategy. Due to the limited knowledge of the detected pages the algorithm has three phases: ascending, middle, and descending phases.

Table 4.2 illustrates the SHARC-Online strategy for the running example. The SHARC-Online crawl starts with p_0 page as a seed and the estimated number of pages in the site $n + 1 = 6$. The crawl downloads page p_0 and detects another two pages p_1, p_2. The algorithm is in its ascending phase, and therefore it schedules the downloads in increasing schedule of the change rate λ_i. In the $I = 2$ iteration the algorithm downloads p_1 and detects additional pages p_3 and p_4. The number of detected and downloaded pages

I	DD-list	
	Downloaded	|Detected
0		$\|p_0$
1	p_0	$\|p_1, p_2$
2	p_0, p_1	$\|p_2, p_3, p_4$
3	p_0, p_1, p_4	$\|p_2, p_3$
4	p_0, p_1, p_4, p_3	$\|p_2$
5	p_0, p_1, p_4, p_3, p_2	$\|p_5$
6	$p_0, p_1, p_4, p_3, p_2, p_5$ |	

Table 4.2.: SHARC-Online Crawl Strategy

exceeds the middle of the interval and the algorithm switches to the middle phase to preserve the middle of the organ-pipes arrangement. The algorithm downloads p_4 and p_3 in the middle phase. Then the number of downloaded pages exceeds the middle the algorithm finishes in the descending phase with the downloads of p_2 and p_5.

4.4.3. Formalization of SHARC-Online

The SHARC-Online strategy maintains the list of detected pages $(p_0^E, p_1^E, \ldots, p_{n^E-1}^E)$ (sorted in ascending order according to the change rates), the number of downloaded pages n^D, the number of detected pages n^E, and an approximated overall number of the pages $n+1$. The SHARC-Online strategy expresses the next page to be downloaded $p_{n^D}^D$ in terms of these three variables.

Ascending Phase. The ascending phase resembles the beginning of the organ-pipes and is applied when the number of downloaded and detected pages is below the estimated middle point of the crawl. During this phase the algorithm implements the coldest-first strategy. Equation (4.12) formalizes the ascending strategy.

$$p_{n^D}^D = p_0^E. \tag{4.12}$$

The ascending strategy is executed as long as the number of downloaded and detected pages is less than half of the size of the site:

$$n^D + n^E \leq \frac{n+1}{2}. \tag{4.13}$$

Example 4.3 (ASCENDING PHASE.) *Consider $I = 1$ step in Table 4.2. The number of downloaded pages $n^D = 1$, the number of detected pages $n^E = 2$, and the list of detected pages sorted in ascending order according to the λs is $(p_0^E, p_1^E) = (p_1, p_2)$ Let us assume that the estimated number of pages in the crawl is $n+1 = 6$. Since $n^D + n^E = 1+2 \leq 3 = \frac{n+1}{2}$, therefore the algorithm is in the ascending phase and the next download element is $p_1^D = p_1^E = p_2$.*

Middle Phase. The middle phase schedules the next download so the symmetry around the middle of the organ-pipes is preserved as much as possible. For each downloaded page on the ascending part, we reserve an appropriate page on the descending part of the organ-pipes. The strategy is applied when the overall number of downloaded and detected pages exceeds the half of the number of the pages, but the number of downloaded pages has not yet reached the middle of the crawl. Equation (4.14) formalizes the phase:

$$p_{n^D}^D = \begin{cases} p_{n^D}^E & \text{if } n^D < n^E, \\ p_{n^E-1}^E & \text{otherwise.} \end{cases} \tag{4.14}$$

Equation (4.15) formalizes the conditions when the middle phase is applied:

$$n^D + n^E > \frac{n+1}{2}, \qquad n^D \leq \frac{n+1}{2}. \tag{4.15}$$

Example 4.4 (MIDDLE PHASE.) *Let us continue Example 4.3 with step $I = 2$. The number of downloaded pages $n^D = 2$, the number of detected pages $n^E = 3$, and the list of detected pages sorted in ascending order according to the λs is:*

$$(p_0^E, p_1^E, p_2^E) = (p_2, p_3, p_4).$$

Since

$$n^D + n^E = 2+3 > 3 = \frac{n+1}{2} \text{ and } n^D = 2 < 3 = \frac{n+1}{2},$$

the algorithm is in its middle phase and the next download element is

$$p_2^D = p_2^E = p_4.$$

Descending Phase. The descending phase resembles the ending of the organ-pipes and is applied when the number of downloaded pages is more than the half of the (estimated) number of pages. During this phase, the algorithm implements the hottest-first strategy. Equation (4.16) formalizes the descending strategy:

$$p_{n^D}^D = p_{n^E-1}^E. \tag{4.16}$$

The descending phase is executed as soon as the number of downloaded pages exceeds the middle of the organ-pipes arrangement and until all detected URLs are downloaded:

$$n^D > \frac{n+1}{2}, \qquad n^E \neq 0. \tag{4.17}$$

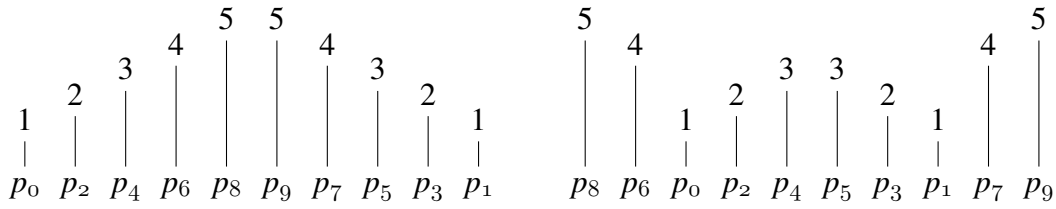
Example 4.5 (Descending Phase) *Let us continue Example 4.4 with step $I = 4$. The number of downloaded pages $n^D = 4$ and the number of detected pages $n^E = 1$. Since $n^D = 4 > 3 = \frac{n+1}{2}$, the algorithm is in its descending phase and the next download element is $p^D_5 = p^E_0 = p_2$.*

4.4.4. SHARC-Online Algorithm

Algorithm 2 depicts the SHARC-Online algorithm. At each iteration the algorithm inspects the sizes of downloaded and detected lists, identifies whether the algorithm is in the ascending, middle or descending phase, and computes the index of the page for the next download. The computation of the index is a direct match to Equations 4.12, 4.14, and 4.16.

Algorithm 2 SHARC-Online

Require: sorted seeds $(p_0, \ldots, p_m)$, estimated size of the crawl n
Ensure: download schedule $(p^D_0, \ldots, p^D_n)$
  $\mathbf{P}^D = (p^D_0, \ldots, p^D_{n^D}) = ()$, $n^D = 0$
  $\mathbf{P}^E = (p^E_0, \ldots, p^E_{n^E}) = (p_0, \ldots, p_m), n^E = m$
  while $\mathbf{P}^E \neq \emptyset$ **do**
    if $n^D + n^E \leq (n+1)/2$ **then**
      $pos = 0$
    else if $n^D \leq (n+1)/2$ **then**
      $pos = n^D < n^E ? n^D : n^E - 1$
    else
      $pos = n^E$
    end if
    append($\mathbf{P}^D, p^E_{pos}$), remove($\mathbf{P}^E, p^E_{pos}$)
    add($\mathbf{P}^E$,urls(p^E_{pos}))
    $n^D{+}{+}$, $n^E{-}{-}$, $n^E = n^E + |\text{urls}(p^E_{pos})|$
  end while



(a) SHARC-Offline (b) Worst Case with $k = 4$

Figure 4.3.: Example of Worst Case Blur Scenario with $n = 9$ and $k = 4$

4.4.5. Worst Case Analysis

In this section we investigate in detail the increase of the blur of the SHARC-Online strategy. According to Theorem 4.1 the increase of the blur of SHARC-Online compared to SHARC-Offline can not exceed more than 100%. In order to get tighter bounds, we analyze the worst case scenario for SHARC-Online. In SHARC-Offline the hottest pages are scheduled in the middle of the crawl interval. Since SHARC-Online does not possess the full knowledge about the URLs of the site, it may download pages at positions different from those of the SHARC-Offline strategy. To analyze the complexity of the task, we assume that SHARC-Online can schedule $(n+1)-k$ downloads optimally. However, k downloads do not follow the SHARC-Offline (k-misplacements). The worst k-misplacements happen if we placed the k hottest pages in the k outermost positions. Example 4.6 illustrates this case:

Example 4.6 (WORST CASE BLUR.) *Consider a Web site of $n+1 = 10$ pages with $\lambda_0 = \lambda_1 = 1$, $\lambda_2 = \lambda_3 = 2$, $\lambda_4 = \lambda_5 = 3$, $\lambda_6 = \lambda_7 = 4$, $\lambda_8 = \lambda_9 = 5$. Let $k = 4$ be the number of pages that do not follow the SHARC-Offline strategy. The optimal SHARC-Offline strategy of this site is illustrated in Figure 4.3a with the worst case scenario in Figure 4.3b.*

The highest schedule penalty positions in the crawl are the first and the last download slots: $\omega(0,10,1) = \omega(9,10,1) = 40.5$, and downloads of the hottest pages (p_8 and p_9 with $\lambda_8 = \lambda_9 = 5$) at these positions maximizes the blur of the archive. The next two highest schedule penalty positions are $\omega(1,10,1) = \omega(8,10,1) = 32.5$ and the next hottest pages p_6 and p_7 are scheduled there. The remaining positions are scheduled according to SHARC-Offline strategy resulting in an organ-pipes-like middle part of the download schedule.

The increase of the blur for the worst case scenario is

$$2(5-1)\omega(0,10,1)+2(4-2)\omega(1,10,1)-2(3-1)\omega(2,10,1) \\ -2(4-2)\omega(3,10,1)-2(5-3)\omega(4,10,1)=145.$$

Since now the blur of the SHARC-Offline is $2\cdot\sum_{i=0}^{4} i(i)=755$*, the relative increase of the blur of the worst case is* $145/755\approx 20\%$.

The following theorem states the increased blur of the worst case scenario with k-misplacements. For simplicity, we assume that the number of pages $n+1$ and the number of misplacements k are even numbers.

Theorem 4.5 (WORST CASE INCREASE) *Let the number of pages in the site* $n+1$ *be even with such change rates of the pages:* $\lambda_0=\lambda_1\leq\lambda_2=\lambda_3\leq\cdots\leq\lambda_{n-1}=\lambda_n$*. Let* k *be the even number of pages that can be misplaced in the optimal SHARC-Offline strategy and* Δ *be the delay between two downloads. In the worst case the blur of the crawl increases by:*

$$2\sum_{i=0}^{k/2-1}(\lambda_{n-2i}-\lambda_{2i})\omega(i\Delta,n,\Delta)-2\sum_{i=0}^{(n+1-k)/2}(\lambda_{k+2i}-\lambda_{2i})\omega\Big(\big(\frac{k}{2}+i\big)\Delta,n,\Delta\Big). \quad (4.18)$$

Proof of Theorem 4.5

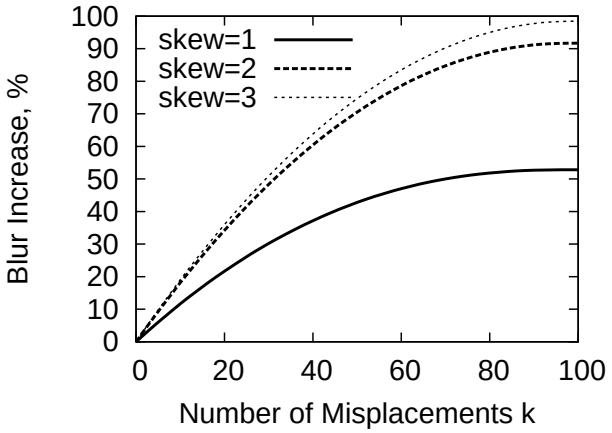
The proof follows from similar arguments as in Theorem 4.3. □

The actual increase in the blur that SHARC-Online exhibits over SHARC-Offline depends on the characteristics of the Web site, especially on the size of the site (number of pages) and the skew in the distribution of the pages' change rates.

Skew has the larger impact, as illustrated in Figure 4.4a. Here we modeled the skew with $\lambda_{2i-1}=\lambda_{2i}=100/i^{skew}$, $i=0,\ldots,n/2$, $n=101$. The increase of the skew by one increases the blur by an order of magnitude. The misplacement of the hottest pages incurs the highest amount of additional blur (see the steep increase for k =0–10 in Figure 4.4a). As the more and more pages are misplaced ($k=20,\ldots,100$) the increase slows down.

Increasing the size $n+1$ of the Web site, illustrated in Figure 4.4b, leads to high additional blur for small n (cf. $n=10^1$ and $n=10^2$ in the figure), while further increasing the number of pages changes the resulting blur only slightly ($n=10^2$ and $n=10^3$ in the figure).

In the next section we compare SHARC-Online against other crawling strategies on both real-world and synthetic datasets.



(a) Skew Varies

	n		
k	10^1	10^2	10^3
00%	00%	00%	00%
20%	28%	34%	36%
40%	48%	60%	63%
60%	60%	79%	83%
80%	64%	89%	95%
100 %	64%	92%	99%

(b) n Varies

Figure 4.4.: Worst Case Blur for Different Values of Skew and Size

4.5. Experimental Evaluation

We evaluated the SHARC-Online against the SHARC-Offline baseline and a selected set of competitors. We first present the competitors under comparison in Section 4.5.1. The experiments were designed so that we could use the exact history of changes as a reference for computing the actual blur of the captures. This metric is defined in Section 4.5.2. Datasets on which the SHARC-Online is evaluated are described in Section 4.5.3. Our main experimental findings on the blur are presented in Section 4.5.4. Finally, we present sensitivity studies in Section 4.5.5 where we vary the Web site properties like the size, change skew, and crawl duration.

4.5.1. Methods under Comparison

We experimentally evaluate our own techniques — SHARC-Offline, SHARC-Online — against a variety of baseline strategies: Breadth-first search (BFS) and Depth-first search (DFS) (most typical techniques by archive crawlers), Hottest-first (HF), Hottest-last (HL) (most promising simple crawlers, where heat refers to change rates of pages), and the method of Olston and Pandey (OP) [75] (the best freshness-optimized crawling strategy). SHARC-Offline requires the knowledge of all URLs of the site in advance. The setup of our experiments allowed us to study this idealized strategy as a reference. In practice, it is unrealistic to assume such knowledge, and SHARC-Online will be used instead. The BFS and DFS strategies schedule downloads based on the graph structure of the site

as it is dynamically traversed by the crawler (the crawl tree). BFS stores the detected URLs in a FIFO queue, while DFS stores the detected URLs in a stack. HF and HL download the hottest and the coldest page from the list of detected pages. All online strategies (SHARC-Online, BFS, DFS, HF, HL) are dynamic and work incrementally: in each iteration we schedule only one page (from among the so far detected pages). The OP strategy sorts the pages in each iteration according to the values of the utility function $U_{p_i}(i)$ and downloads the one with the highest value. The utility function $U_{p_i}(i)$ assumes knowledge of the full change history. This assumption is not practical. We give these optimistic performance numbers for comparison.

For incorporating change rate estimates, we include two versions: the change rate is given either by the oracle or by a predictor. Chapter 6 describes in detail the predictors and the features used for training. A random sample of 10% of the size of each Web site was used to train the predictors.

4.5.2. Quality Metrics

We use the actual blur (Equation (4.19)) to assess single-visit strategies.

The intuition behind the actual blur is the same as the one behind the stochastic blur (Definitions 4.1: the average number of changes that an explorer of the archive will encounter. The stochastic blur uses change rates λ_i while the actual blur uses the history of actual timepoints of changes $(h_0,\ldots,h_m)$. In combination with the download time t of page p and the observation interval $[o_s,o_e]$, this allows us to define the actual blur of p:

$$B(p) = \frac{1}{o_e - o_s}\Big(\sum_{o_s \le h_j \le t} (h_j - o_s) + \sum_{t < h_j \le o_e} (o_e - h_j)\Big). \tag{4.19}$$

The actual blur B of an entire site capture with pages $(p_0,\ldots,p_n)$ is the sum of the actual blur values of all pages:

$$B(p_0,\ldots,p_n) = \sum_{i=0}^{n} B(p_i).$$

The actual blur with revisits can be defined similarly. Let t^v be the visit and t^r be the revisit time of page p. Then the *actual blur with revisits* for observation interval $[o_s,o_e]$

is:

$$B(p) = \frac{1}{o_e - o_s}\Big(\sum_{o_s < h_j \le t^v} (h_j - o_s) + \sum_{t^v < h_j \le \frac{t^v + t^r}{2}} (\frac{t^v + t^r}{2} - h_j) + \sum_{\frac{t^v + t^r}{2} < h_j \le t^r} (h_j - \frac{t^v + t^r}{2} + \sum_{t^r < h_j \le o_e} (o_e - h_j)\Big). \quad (4.20)$$

The intuition behind this formula is that for each page that should be accessed as of observation time t, we can choose either the version as of the visit time or the version as of the revisit time, whichever is closer to t.

The actual blur with revisits of a site capture with pages $(p_0, \ldots, p_n)$ is the sum of the corresponding values of all pages:

$$B(p_0, \ldots, p_n) = \sum_{i=0}^{n} B(p_i).$$

The OP utility function U_p can be expressed in terms of actual blur. U_p is the actual blur of page p in interval $[o_s, o_e]$ given that the page is downloaded at o_e: $U_p = B(p) = \frac{1}{o_e - o_s} \sum_{j=0}^{m} (h_j - o_s)$. The utility function gives a higher priority to pages with late changes.

4.5.3. Datasets

We tested our methods on the real-world datasets introduced in Chapter 3 and also on synthetically generated Web sites for systematic variation of site properties.

The real-world datasets consist of Web archives of an academic Web site (MPII), a Web directory (DMOZ) and five governmental Web sites (UKGOV). We split the UKGOV dataset into five parts: MOD, DFID, ARMY, RAF, and DH. Recalling the information in Table 3.1, DMOZ is the largest dataset with over 177 000 Web pages, while DH is the smallest with around 2 000 Web pages. The MPII dataset consists of daily crawls, the other datasets of weekly crawls. They are used for our main experiments about blur, presented in Section 4.5.4.

We used the available datasets to simulate crawls and evaluate the SHARC-Offline and SHARC-Online strategies. As a stress test for politeness-aware archive crawling, we artificially slowed down the crawls so that they would take as long as the entire time period covered by the dataset. This is done by replaying crawls from our stored data with virtual time. Obviously, slowing down a crawl so that it would take months or even a

whole year is quite extreme and does not correspond to what you would do in reality. But this way we impose a large number of page changes on each crawl and turn our experiments into more informative stress-test studies.

The synthetic datasets simulate changes according to the Poisson process and are used in sensitivity experiments (see Section 4.5.5). The change rates are modeled with a skewed distribution: $\lambda_i = 1/((i+1)^{\text{skew}})$. The Web graph of a synthetic site is generated to form a tree with n pages; each page p_i has outdegree children. The default values are skew $= 1.2$, outdegree $= 400$, and $n = 10,000$. To correlate site structure with skewed change rates, we generated Web sites in two flavors. The first flavor has the the hottest page at the root of the site tree, and the pages are gradually "cooling down" towards the leaves (denoted by "Cold Leaves" in Tables 4.4–4.6). The second flavor consists of sites where the root is coldest, and the pages are gradually "heating up" towards the leaves (denoted by "Hot Leaves" in the captions).

4.5.4. Blur Experiments with Real-World Datasets

The results of the experiments on blur with single-visit strategies over real-world datasets are shown in Table 4.3. The best values for each dataset are highlighted in boldface. SHARC-Offline outperforms all competitors by a large margin. SHARC-Online with change-rate oracle performs nearly as well as the optimal SHARC-Offline method. Its additional burden, compared to SHARC-Offline, is that it needs to incrementally detect the pages of a Web site. The experiments show that the penalty of this page-discovery process is low.

SHARC-Online with predicted change rates leads a more pronounced increase of the blur metric. Obviously, estimation errors about which pages are hot and which ones are not so hot have a notable influence on the overall quality of a site capture. As a consequence, SHARC-Online may even be slightly inferior to more traditional baselines on some data, but it wins by a large margin for most datasets. Note that the traditional baselines used a change-rate oracle; so the slight losses of SHARC-Online are mostly due to the facts that the opponents had better a priori knowledge about changes.

Overall, this experiment showed that our blur-optimizing strategies do indeed provide what our theory suggested: they clearly improve the quality of site captures for web archiving.

	SHARC			HF	HL	BFS	DFS	OP
	Offline	Online	Online					
Site	Oracle	Oracle	Predicted	Oracle	Oracle			Oracle
MPII	**0.12**	0.13	0.16	0.26	0.15	0.24	0.15	0.18
DMOZ	**0.18**	0.19	0.23	0.24	0.31	0.25	0.25	0.24
MOD	**2.14**	2.16	2.17	2.48	2.98	2.41	2.46	3.15
DFID	**2.11**	2.17	2.17	3.35	2.20	2.51	2.19	2.17
ARMY	**1.23**	1.26	1.29	1.55	1.65	1.56	1.45	1.81
RAF	**0.11**	**0.11**	0.14	0.14	0.15	0.14	0.15	0.15
DH	**2.82**	2.83	2.94	4.00	3.08	3.08	3.08	3.27

Table 4.3.: Average Blur per Page

4.5.5. Sensitivity Studies

In this subsection, we studied the sensitivity of our crawl strategies with regard to the scale (size) of a Web site, the skew in the change rate distribution of a site's pages, and the politeness-driven duration of the crawl. As we wanted to vary these parameters systematically, we performed these experiments with synthetically generated Web sites.

Scalability. Tables 4.4a and 4.4b show blur results for Web sites with hotter pages closer to the root and hotter pages closer to the leaves, respectively. While SHARC-Online is only slightly worse than SHARC-Offline in Table 4.4b, the difference between the strategies is more prominent in Table 4.4a. This difference appears because we discover cold pages much later in Table 4.4b, and misguide the schedule for page downloads by seeing mostly hot pages early on.

	SHARC		HF	HL	BFS	DFS	OP
Size	Offline	Online					
$1 \cdot 10^4$	**1.21**	1.70	1.75	2.26	2.00	1.80	2.28
$5 \cdot 10^4$	**1.25**	1.75	1.80	2.38	2.07	1.83	2.42
$1 \cdot 10^5$	**1.27**	1.77	1.82	2.44	2.13	1.87	2.48
$5 \cdot 10^5$	**1.30**	1.81	1.86	2.52	2.18	2.00	2.56
$1 \cdot 10^6$	**1.31**	1.83	1.87	2.55	2.19	2.01	2.59

(a) Cold Leaves

	SHARC		HF	HL	BFS	DFS	OP
Size	Offline	Online					
$1 \cdot 10^4$	**1.19**	1.21	2.24	1.44	1.28	1.22	2.25
$5 \cdot 10^4$	**1.25**	1.26	2.39	1.53	1.92	1.81	2.42
$1 \cdot 10^5$	**1.26**	1.27	2.43	1.55	1.36	1.30	2.47
$6 \cdot 10^5$	**1.29**	1.30	2.52	1.60	1.39	1.96	2.56
$1 \cdot 10^6$	**1.31**	**1.31**	2.55	1.61	2.21	1.94	2.59

(a) Hot Leaves

Table 4.4.: Scalability: Average Blur per Page

Slow-down	SHARC Offline	SHARC Online	HF	HL	BFS	DFS	OP
1	**1.20**	1.68	1.97	2.25	2.21	1.78	2.28
2	**2.41**	3.38	3.96	4.50	4.36	3.46	4.41
3	**3.58**	5.02	5.89	6.70	6.48	5.07	6.54
4	**4.79**	6.72	7.88	8.98	8.81	7.28	8.77
5	**5.99**	8.38	9.83	11.21	10.77	8.58	10.95
10	**12.00**	16.83	19.75	22.48	21.62	16.75	22.00

(a) Cold Leaves

Slow-down	SHARC Offline	SHARC Online	HF	HL	BFS	DFS	OP
1	**1.20**	1.22	2.27	2.10	1.86	1.85	2.27
2	**2.39**	2.42	4.48	4.17	2.47	2.46	4.40
3	**3.60**	3.65	6.73	6.25	3.92	4.02	6.56
4	**4.80**	4.86	8.99	8.37	7.55	7.47	8.77
5	**5.98**	6.07	11.19	10.42	8.29	8.24	10.94
10	**12.01**	12.18	22.47	20.92	17.67	18.79	22.03

(a) Hot Leaves

Table 4.5.: Crawl Duration: Average Blur per Page

SHARC-Online consistently outperforms all baseline opponents. Schedules of HF for the dataset with cold leaves and HL for the hot leaves are very similar to the schedule of SHARC-Online. Consequently, HF and HL perform almost as well as SHARC-Online in some cases, but strongly deteriorate in the other cases. Moreover, they are much more sensitive to the size of a Web site. The sensitivity comes from the strong influence of the different sizes on the number of leaves and consequently on the placement of the hot and cold pages in the schedule.

Crawl Duration. In this experiment we increased the politeness delay by a specified slow-down factor. Tables 4.5a and 4.5b show the resulting blur values. For all strategies the capture interval increases, and in turn, the blur increases as well. For short captures all competitors perform similarly, as crawling is almost "instantaneous". For longer captures, our SHARC strategies become increasingly advantageous over the competitors.

SHARC-Online outperforms all other online strategies. The only exception is the DFS strategy for the longest crawl and the dataset with cold leaves. The specific placement of the changes in the tree of the Web site made the schedule of the DFS slightly closer to the schedule of SHARC-Offline. However, as the placement of the hot pages changes (Table 4.5b) the cold pages are discovered earlier and SHARC-Online substantially outperforms DFS.

Skew. Skew controls how uniformly the changes are distributed among the pages. High skew allocates most of the changes to very few pages, while low skew keeps the changes uniformly distributed. In absolute numbers, this results in high blur for low skew and low blur for high skew for all strategies. Table 4.6a shows the results for this sensitivity study with the single-visit strategies aiming at low blur. Relative to the baseline

opponents, the SHARC strategies cope best with high skew. Their relative gains increase with increasing skew.

	SHARC		HF	HL	BFS	DFS	OP
Skew	Offline	Online					
0.50	**60.49**	67.00	73.53	73.84	71.79	67.63	69.23
1.00	**2.57**	3.35	3.99	4.40	4.29	3.54	4.20
1.50	**0.61**	0.88	1.08	1.21	1.18	1.00	1.22
2.00	**0.37**	0.58	0.70	0.74	0.74	0.63	0.74

(a) Cold Leaves

	SHARC		HF	HL	BFS	DFS	OP
Skew	Offline	Online					
0.50	**60.38**	62.38	73.71	72.02	69.01	69.56	69.23
1.00	**2.55**	2.61	4.39	4.17	3.10	3.15	4.16
1.50	**0.62**	**0.62**	1.22	1.16	1.06	1.11	1.22
2.00	**0.38**	**0.38**	0.75	0.71	**0.38**	**0.38**	0.75

(b) Hot Leaves

Table 4.6.: Skew: Average Blur per Page

4.6. Summary

In this chapter we defined the blur quality measure for Web archives. It is a stochastic measure appropriate for explorative use of Web archives. The properties of the blur allowed us to design SHARC-Offline — a theoretically optimal crawling strategy. Since the strategy requires full knowledge of the Web site in advance which is not always available, we introduced SHARC-Online — a crawling strategy which resembles SHARC-Offline but learns the Web site on-the-fly. We investigated the factors which influence the performance of the crawling strategies. Furthermore, we showed that the both of the strategies outperform the established crawling strategies in terms of quality of the resulting captures. Since the blur measure is stochastic, it does not give deterministic guarantees on the Web archive quality. For that purpose, in the next chapter, we define coherence - a deterministic measure quality measure and develop the visit-revisit strategies which maximize the coherence.

Chapter 5.

Visit Revisit Crawling Strategies

5.1. Introduction

In this chapter we investigate the crawling strategies which tell exactly which pages changed. These strategies download each page twice, where the second download (the revisit) serves to check for changes. All the revisits follow all the visits. This approach allows us to reason deterministically about the state of the archive as of the middle point of the crawl, which we call the *reference timepoint*. The *coherence* of an entire site capture at the reference time pint is the number of coherent pages. A page is *coherent* if its versions at visit time and revisit time are identical.

Definition 5.1 (COHERENCE.) *A page in a site capture is coherent if it did not change between its visit and revisit. The coherence of a site capture is the number of pages in the archived capture that did not change between their visit and revisit timepoints.*

We investigate three approaches for visit-revisit crawling. First, the archive crawls for the least blurred capture. Second, the archive crawls for the capture with maximum hopeful pages. Finally, the archive crawls for the capture with highest expected coherence. In the next sections we cover the each approach, its underlying model and respective download strategy or strategies. Section 5.2 defines blur for a capture with revisits and presents the SHARC-Revisits download strategy which minimizes the blur. Section 5.3 introduces the concepts of *hopeful* and *hopeless* pages as well as the SHARC-Threshold strategy which maximizes the number of hopeful pages in a download schedule. Section 5.4 presents the SHARC-Intervals strategy which is an alternative to SHARC-Threshold. It makes use of confidence intervals to increase the number of hopeful pages. Section 5.5 formally defines the expected coherence of a schedule and develops the SHARC-Selective

download strategy which maximizes the expected coherence.

5.2. SHARC-Revisits

In this section we extend the definition of blur for captures with vistis and revisits.

Given Web pages $p_0, p_1, \ldots, p_n$ and the change rates $\lambda_0 \leq \lambda_1 \leq \cdots \leq \lambda_n$ the task is to find the timepoints (download slots) for the initial visits $t_0^v, t_1^v, \ldots, t_n^v$ and for the revisits $t_0^r, t_1^r, \ldots, t_n^r$ such that the blur of the site capture is minimized. Since the archive now consists of two versions of the page we return the version of the page that is closer to the given query time t (cf. $\min\{|t_i^v - t|, |t_i^r - t|\}$ in the definition below).

Definition 5.2 (BLUR OF PAGE WITH REVISITS.) *Let p_i be a Web page with visit time t_i^v and revisit time t_i^r. The* blur of page p_i *is*

$$\begin{aligned} B(p_i, t_i^v, t_i^r, n, \Delta) &= \frac{1}{(2n+1)\Delta} \int_0^{2n\Delta+1} \lambda_i \min\{|t_i^v - t|, |t_i^r - t|\} \\ &= \frac{1}{(2n+1)\Delta} \lambda_i \Big(\int_0^{(t_i^v+t_i^r)/2} |t_i^v - t| \, dt + \int_{(t_i^v+t_i^r)/2}^{(2n+1)\Delta} |t_i^r - t| \, dt \Big) \\ &= \frac{1}{(2n+1)\Delta} \lambda_i \omega(t_i^v, t_i^r, n, \Delta), \end{aligned} \tag{5.1}$$

where

$$\omega(t_i^v, t_i^r, n, \Delta) \;=\; (t_i^v)^2 \;-\; \frac{(t_i^v + t_i^r)^2}{4} \;+\; (t_i^r)^2 \;-\; t_i^r(2n \;+\; 1)\Delta \;+\; \frac{(2n+1)^2\Delta^2}{2} \tag{5.2}$$

is the download schedule penalty for downloads with revisits. The blur of an archived capture *with revisits is the sum of the blur values of the individual pages:*

$$B(P, T^v, T^r, n, \Delta) = \sum_{i=0}^{n} B(p_i, t_i^v, t_i^r, n, \Delta), \tag{5.3}$$

where $T^v = (t_0^v, \ldots, t_n^v)$ are the visit and $T^r = (t_0^r, \ldots, t_n^r)$ are the revisit times of pages $P = (p_0, \ldots, p_n)$. The average blur is $\bar{B}(P, T^v, T^r, n, \Delta) = 1/n \sum_{i=0}^{n} B(p_i, t_i^v, t_i^r, n, \Delta)$.

Example 5.1 (BLUR WITH REVISITS.) *Consider the Web site in Figure 4.1 with $t_0^v = 0, t_1^v = 1, \ldots, t_5^v = 5$ visit and $t_0^r = 6, t_1^r = 7, \ldots, t_{11}^r = 11$ revisit times. The blur of page p_1*

is

$$B(p_1,1,7,5,1) = \frac{1\cdot\omega(1,7,5,1)}{2\cdot 5+1} = 1^2 - \frac{(1+7)^2}{4} + 7^2 - 7(2\cdot 5+1) + \frac{(2\cdot 5+1)^2}{2} = 35/22. \quad (5.4)$$

Similarly, $B(p_0,0,6,5,1) = 0$, $B(p_2,2,8,5,1) = 62/22$, $B(p_3,3,9,5,1) = 93/22$, $B(p_4,4,10,5,1) = 140/22$, $B(p_5,5,11,5,1) = 215/22$, *and the blur of the archive is*

$$B(P,T^v,T^r,4,1) = 0 + \frac{35}{22} + \frac{62}{22} + \frac{93}{22} + \frac{140}{22} + \frac{215}{22} \approx 24.77.$$

The derivation of the optimal visits $t_0^v,\ldots,t_n^v$ and revisits $t_0^r,\ldots,t_n^r$ for minimum blur is similar to the analysis of the optimal download schedule without revisits (cf. Theorem 4.3).

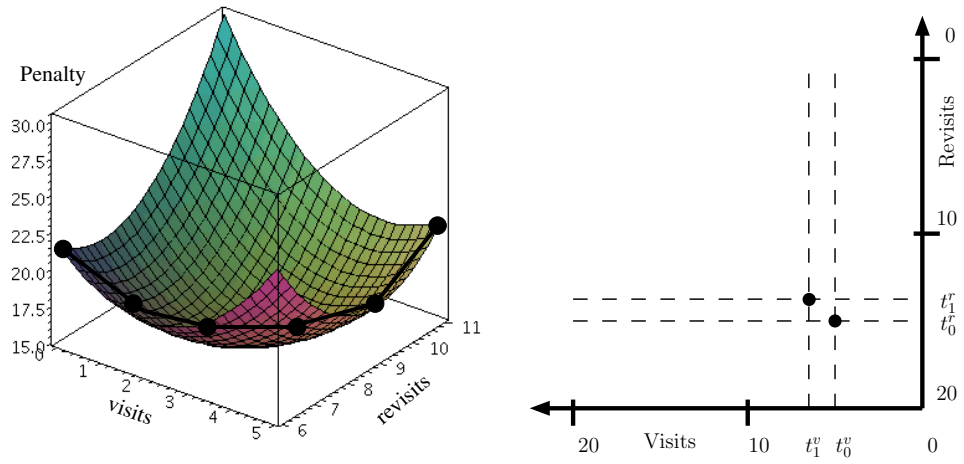
Again, we need to schedule pages in ascending order of λ values and descending order of download schedule penalties $\omega(p_i,t_i^v,t_i^r,n,\Delta)$, so that the product of these factors minimizes the overall sum in Equation 5.1. Similarly to the download schedule penalty without revisits, the penalty with revisits is always an elliptic paraboloid w.r.t. t_i^v,t_i^r, with one minimum (cf. Figure 5.1a). Equation 5.1 suggests the following strategy towards minimizing blur. We schedule the visit and revisit of the hottest page in the download slots with the smallest penalty (t_0^v,t_0^r) (cf. Figure 5.1b). Then we mark all points (t_0^v,t) and (s,t_0^r) as invalid and search the next valid position with smallest penalty, and so on. This strategy results in visit-revisits forming a diagonal line in the visit-revisit plane (cf. filled circles in Figure 5.1a).

The strategy is greedy: at each step we aim to assign the hottest change rate at the lowest penalty position. While the strategy yielded an optimum in the single-visit case, this is not necessarily the optimum for the visit-revisit case. To obtain an optimum schedule we would need to scan all possible parabola of visits-revisits in the elliptic paraboloid and check for which the sum of the factors of change rates and penalty positions yield the smallest blur.

Definition 5.3 (SHARC-REVISITS.) *Let* $P = (p_0,\ldots,p_n)$ *be the Web site such that* $\lambda_0 \le \cdots \le \lambda_n$. *The pair*

$$(t_i^v,t_i^r) = \begin{cases} (\frac{i}{2}, n+1+\frac{i}{2}) & \text{if } i \text{ is even and} \\ (n-\frac{i-1}{2}, 2n+1-\frac{i-1}{2}) & \text{otherwise} \end{cases}$$

defines the greedy strategy *for the visit and revisit times of page* p_i.



(a) Penalty (b) Greedy Approach

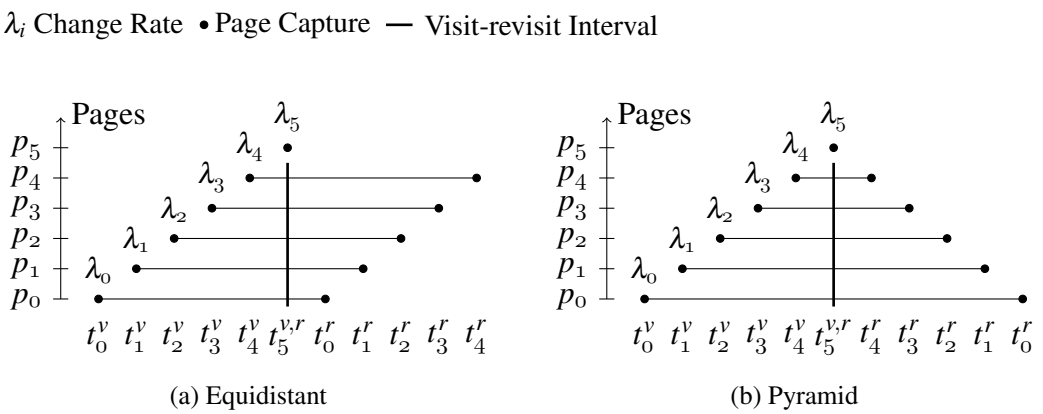
Figure 5.1.: Optimization of Crawling with Revisits

Example 5.2 (SHARC-REVISITS) *Let us continue Example 5.1. The greedy visit and revisit times for pages* $p_0, \ldots, p_5$ *are* $(t_0^v, t_0^r) = (0, 6)$, $(t_1^v, t_1^r) = (5, 11)$, $(t_2^v, t_2^r) = (1, 7)$, $(t_3^v, t_3^r) = (4, 10)$, $(t_4^v, t_4^r) = (2, 8)$, $(t_5^v, t_5^r) = (3, 9)$. *The blur of the greedy schedule is approximately 22.59.*

5.3. SHARC-Threshold

Visit-revisit strategies schedule all visits before the revisits so that intervals between visit and revisit have a non-empty intersection. With this approach, the ideal outcome would be that all pages are mutually coherent if they individually did not change between their visits and revisits. Section 5.2 developed a strategy which minimized the blur but did not maximize the coherence. In this section we are looking for a strategy which schedules the visits and the revisits in such way that the coherence increases. This is equivalent to find the best order of visit-revisit intervals.

Two extreme choices of visit-revisit intervals are equidistant schedule where all intervals have the same lengths, as shown in Figure 5.2a, and pyramid-like schedule, shown in Figure 5.2b, where the intervals are centered around a the reference timepoint. Allocation of pages (change rates) to the intervals is the degree of freedom of the strategies. Intuitively, one could allocate the hottest pages to the shortest intervals greedily maximizing each page expected coherence. However, in certain cases it is better to "give up" extremely hot (*hopeless*) pages, by assigning them to longer visit-revisit intervals so



(a) Equidistant (b) Pyramid

Figure 5.2.: Shapes of Schedules

that other (*hopeful*) pages get shorter visit-revisit intervals and, therefore, have higher chances of getting coherently captured. In the current and in the following sections we investigate two alternative strategies to identify hopeless pages: SHARC-Threshold and SHARC-Intervals. We start with SHARC-Threshold.

Definition 5.4 (SHARC-THRESHOLD STRATEGY) *Let $p_0,\ldots,p_n$ be the Web site with $\lambda_0 \leq \cdots \leq \lambda_n$. Let τ be the threshold parameter. The following defines the visit-revisit intervals iteratively.*

Let p be the hottest page at iteration i such that the probability $P[p$ is coherent in$[(n-i)\Delta,(n+i+1)\Delta]] = 1-\exp\{-(2(n-i)+1)\Delta\lambda\}$ is greater than τ then page p is hopeful and its visit-revisit interval is

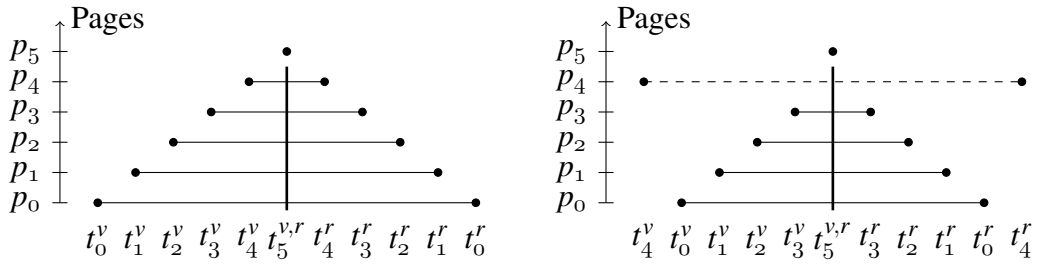
$$[(n-i)\Delta,(n+i+1)\Delta],$$

otherwise the page is declared hopeless and is postponed until the end.

Let $p_0^h,\ldots,p_k^h$ be the set of hopeless pages such that $\lambda_0^h \leq \cdots \leq \lambda_k^h$. Then the visit-revisit interval of p_i^h is $[(k-i)\Delta,(2n-1-k+i)\Delta]$

Theorem 5.1 (AVERAGE NUMBER OF COHERENT PAGES)

Let $p_0^p,\ldots,p_l^p$ and $p_0^h,\ldots,p_k^h$ be the hopeful and hopeless pages. Let the changes of the pages be distributed according to the independent Poisson processes with $\lambda_0^p \leq \cdots \leq \lambda_k^p$ and $\lambda_0^h \leq \cdots \leq \lambda_l^h$. Let $[k+i+1,2l+k+2-i]$ be the visit-revisit interval of p_i^p and $[k-j,2l+k+j+3]$ be the visit-revisit interval of p_j^h of the SHARC-Threshold schedule. Then the average number of coherent pages is



(a) All Pages are Hopeful

(b) p_4 is Hopeless

Figure 5.3.: SHARC-Threshold

$$\sum_{i=0}^{l} e^{-\lambda_i^p \left(2(l-i)+1\right)\Delta} + \sum_{j=0}^{k} e^{-\lambda_j^h \left(2(l+j)+3\right)\Delta}.$$

Proof of Theorem 5.1 *The proof follows from the properties of the Poisson process.* □

Figure 5.3 illustrates the difference between schedules with hopeless pages and schedules without. Consider the pages from Figure 4.1: pages p_0 to p_5 with ascending change rates. Figure 5.3a shows the schedule if none of the pages is hopeless. Each page gets visit-revisit intervals according to its change rate - the higher the change rate, the shorter the interval. Figure 5.3a shows the schedule if p_4 is hopeless. In that case p_4 gets the longest interval. In return, the intervals of the other pages get shorter.

To apply the SHARC-Threshold in practice, we have to choose an appropriate value for the threshold parameter τ. Intuitively, the threshold must be high enough to guarantee that hopeful pages are indeed coherent. Following this argument, we can set $\tau = 0.95$. Experiments with synthetic datasets with varying change rate distributions (cf. Table 5.1) show that for skewed distributions the best value for τ is 0.95. In datasets with less skewed change rate distributions, a low value for τ achieves better results. Under the assumption that Web sites resemble datasets with skewed distribution of change rates (few frequently changing pages and lots of static pages), we set $\tau = 0.95$.

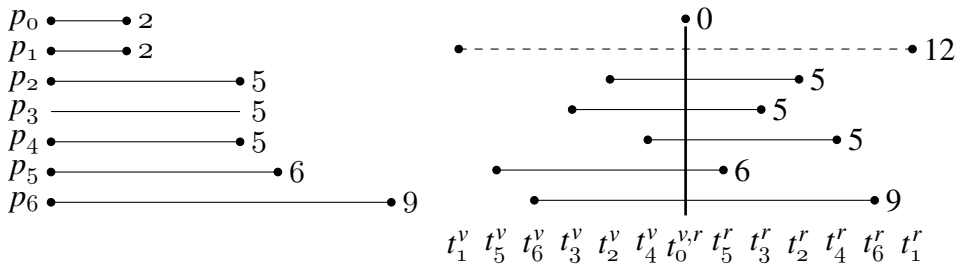
SHARC-Threshold with varying values for τ										
Skew	$\tau = 0.1$	$\tau = 0.2$	$\tau = 0.3$	$\tau = 0.4$	$\tau = 0.5$	$\tau = 0.6$	$\tau = 0.7$	$\tau = 0.8$	$\tau = 0.9$	$\tau = 0.95$
1.00	8619	7439	**7097**	7108	7164	7202	7248	7266	7260	7257
1.50	1291	1239	1188	1125	1045	961	848	723	549	**519**
2.00	235	221	210	197	187	162	144	129	100	**78**

Table 5.1.: SHARC-Threshold Incoherence with Varying τ

5.4. SHARC-Intervals

An alternative to the pyramid-like schedules is a schedule where the visit-revisit intervals are computed in advance for each page based on their change rates. In this chapter we present the SHARC-Intervals strategy which follows the aforementioned principle.

SHARC-Intervals aims to maximize the number of coherent pages in a site capture by judicious scheduling of visit-revisit intervals based on estimated *confidence intervals* for the absence of changes in a page (explained below). Figure 5.4 illustrates this by example. The algorithm considers the lengths $\mathbb{I} = (I_0, \ldots, I_n)$ of the confidence intervals (the numbers on the right end of the intervals in Figure 5.4a) to schedule the visits $t_0^v, \ldots, t_n^v$ and revisits $t_0^r, \ldots, t_n^r$ so that (i) t_i^v and t_i^r get unique positions, except for one page for which visit and revisit collapse onto the same timepoint, (ii) there is at least a minimum politeness delay Δ between any two, and (iii) the pair t_i^v, t_i^r forms an interval of length less than or equal to I_i. An equivalence relation between scheduling of pages and scheduling of confidence intervals immediately follows Condition (iii).



(a) Input (b) Schedule

Figure 5.4.: Scheduling of Confidence Intervals

Conditions (i) and (ii) arise from the politeness etiquette and must be obeyed by any schedule. Violation of Condition (iii) would entail that a page has a high risk of being changed between its visit and revisit. We can always shorten the timespan between visit and revisit (if the other conditions can still be ensured), but should avoid making it too long. We may face a situation, though, where some pages have very short confidence intervals or the appropriate download slots are already used to schedule other pages. Then these pages are identified as *hopeless*, and will arrive at a (still large) fraction of the site for which coherence is statistically guaranteed. Figure 5.4b shows the output by SHARC-Intervals for the input in Figure 5.4a. Here SHARC-Intervals successfully schedules six intervals: five intervals (pages $p_2, \ldots, p_6$) are of maximal length, interval of

page p_0 is shortened, and page p_1 is hopeless.

The lengths I_i of the confidence intervals can be computed by the commonly used Poisson model for changes of the Web. Given the average change rate λ_i of page p_i the probability that page p_i does not change in an interval of length I_i is

$$P(\text{no change of } p_i \text{ in interval of length } I_i) = \frac{e^{-\lambda_i I_i}(\lambda_i I_i)^0}{0!}.$$

If we require that this probability is above a threshold τ (e.g., 90%), then the length of the confidence interval becomes

$$I_i = \lambda_i^{-1} \log \tau^{-1}. \tag{5.5}$$

For brevity, we will omit the word "confidence" and call the confidence intervals just intervals and the lengths of the confidence intervals as just "lengths".

For simplicity, we focus on a discretized version of the problem: we assume that all intervals are of integer lengths ($\mathbb{I} \subset \mathbb{Z}^+ \cup 0$), and the delay between two download positions is exactly Δ. To simplify mathematical equations we set Δ to 1.

5.4.1. SHARC-Intervals Offline Algorithm

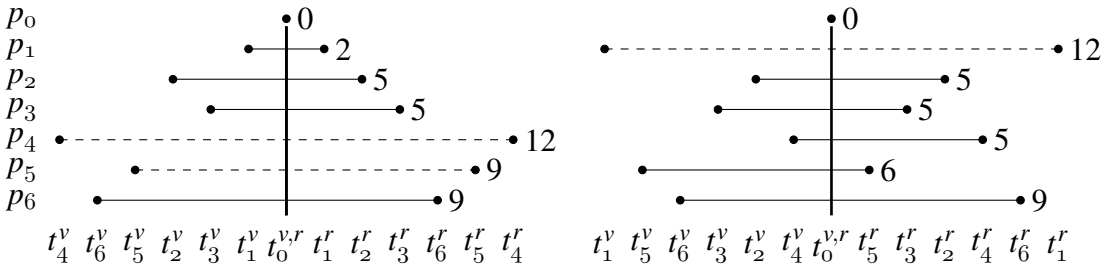
The offline algorithm takes the lengths $\mathbb{I}$ as input and arranges the largest possible number of non-hopeless intervals. The start and end points of the intervals get unique visit and revisit positions. The offline SHARC-Intervals schedules the revisit positions first and the visit positions fall out automatically.

Definition 5.5 (SCHEDULE, HOPELESS AND HOPEFUL PAGES.) *Let $\mathbb{I} = (I_0, \ldots, I_n)$ be the lengths of intervals. Let $\mathbb{V} = (t_0^v, \ldots, t_n^v)$ $(i = 0, \ldots, n)$ be the positions of visits and $\mathbb{R} = (t_0^r, \ldots, t_n^r)$ $(i = 0, \ldots, n)$ be the positions of revisits such that: (i) all download positions are unique: $t_i^v \neq t_j^v$ for all $i \neq j$, $i, j \in \{0, \ldots, n\}$; $t_i^v \neq t_j^r$ for all $i, j \in \{0, \ldots, n\}$ except one pair (t_l^v, t_l^r), for which $t_l^v = t_l^r = n$, and (ii) all intervals are scheduled: $\cup_i \{t_i^v\} = \{0, 1, \ldots, n\}$ and $\cup_i \{t_i^r\} = \{n, n+1, \ldots, 2n\}$.*

Then $\mathbb{V}$ and $\mathbb{R}$ is a schedule of $\mathbb{I}$. Length I_i in schedule $\mathbb{V}$ and $\mathbb{R}$ is hopeless if the corresponding visit-revisit pair is longer than the length: $t_i^r - t_i^v > I_i$. Otherwise the interval I_i is hopeful. Correspondingly, we call the page p_i hopeless or hopeful too.

Example 5.3 (SCHEDULE, HOPELESS AND HOPEFUL PAGES.) *Assume that the lengths are as in Figure 5.4a. Then $t_0^v = t_0^r = 6$, $t_1^v = 5, t_1^r = 7$,*

$t_2^v = 3, t_2^r = 8$, $t_3^v = 4, t_3^r = 9$ $t_6^v = 1, t_6^r = 10$, *(hopeful pages),* $t_4^v = 2, t_4^r = 11$ *and* $t_5^v = 0, t_5^r = 12$ *(hopeless pages) is a schedule of the intervals. The schedule is illustrated in Figure 5.5a.*



(a) Offline SHARC-Intervals

(b) Alternative Schedule with Minimal Number of Hopeless Pages

Figure 5.5.: SHARC-Intervals Schedules

Given all intervals in advance, the offline SHARC-Intervals algorithm schedules the pages based on the following principles: First, the algorithm is greedy. It schedules short intervals first in ascending order of length, since there is less leeway for them. Second, the scheduling is iterative: given $i-1$ scheduled hopeful pages, the algorithm looks for a visit-revisit interval with maximum length not exceeding the length if the i-th interval. Third, if no visit-revisit pair can be found in the i-th planning step the page is declared hopeless. All hopeless pages are scheduled at the end (after all hopeful pages), in descending order of their interval lengths. Pseudocode for the offline SHARC-Intervals is given in Algorithm 3.

5.4.2. Shrinking a Schedule

During the iterative construction of the offline SHARC-Intervals, we always use the full lengths of the pages' confidence intervals. That is, the lengths are never shortened for possibly achieving higher number of hopeful pages in later iterations. This may produce a much wider schedule than necessary (cf. Figure 5.6), or it may schedule some hopeless pages on visit-revisit positions in between the hopeful ones (cf. page p_6 in Figure 5.6a). In both cases we can improve the schedule. We can exchange the visit positions of hopeful pages with visit positions of hopeless pages if the hopeless pages are visited after the hopeful ones. Thus, the visit-revisit intervals of the hopeful pages get shorter in contrast

Algorithm 3 Offline SHARC-Intervals

Require: lengths sorted descending: $I_0, \ldots, I_n$

Ensure: visit-revisit schedule: $\mathbb{V}$ and $\mathbb{R}$

list of hopeless pages: $\mathbb{H}$

Initialize the schedule $\mathbb{V} = \mathbb{R} = (0)$

Initialize the list of hopeless pages $\mathbb{H} = \emptyset$

Initialize the list of visit positions $\mathbb{L}_\mathbb{V} = \{0, 1, \ldots, n-1\}$

Initialize the list of revisit positions $\mathbb{L}_\mathbb{R} = \{n+1, n+2, \ldots, 2n\}$

for $i = 1, \ldots, n$ **do** ▷ // Identify and schedule hopeful pages

find visit $v \in \mathbb{L}_\mathbb{V}$ and revisit $r \in \mathbb{L}_\mathbb{R}$, such that $r - v \leq |I_i|$ and is maximal

if v and r exist **then**

$\mathbb{V} = \mathbb{V} \oplus v \qquad \mathbb{R} = \mathbb{R} \oplus r$

$\mathbb{L}_\mathbb{V} = \mathbb{L}_\mathbb{V} \setminus v \qquad \mathbb{L}_\mathbb{R} = \mathbb{L}_\mathbb{R} \setminus r$

else

$\mathbb{H} = \mathbb{H} \oplus I_i$

end if

end for

Sort $\mathbb{H}$ ascending ▷ // Schedule hopeless pages

for all $h \in \mathbb{H}$ **do**

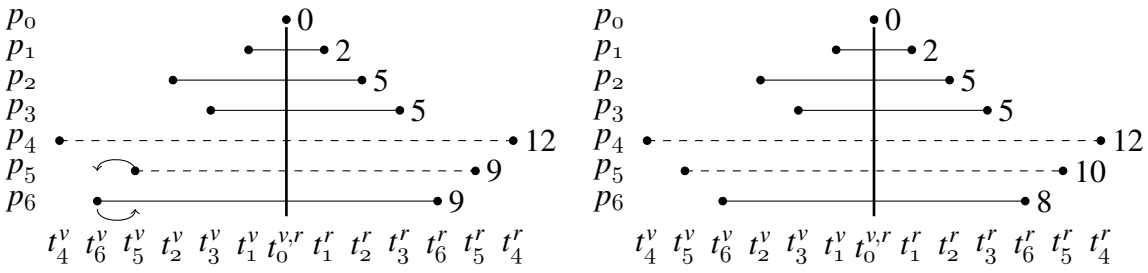
Find largest $v \in= \mathbb{L}_\mathbb{V}$ and smallest $r \in \mathbb{L}_\mathbb{R}$

$\mathbb{V} = \mathbb{V} \oplus v \qquad \mathbb{R} = \mathbb{R} \oplus r$

$\mathbb{L}_\mathbb{V} = \mathbb{L}_\mathbb{V} \setminus v \qquad \mathbb{L}_\mathbb{R} = \mathbb{L}_\mathbb{R} \setminus r$

end for

to the visit-revisit intervals of the hopeless pages which get longer. This increases the chance for coherence of the hopeful pages.



(a) Shrinking the Offline SHARC-Intervals (b) Shrunk Download Schedule

Figure 5.6.: Shortening of Intervals

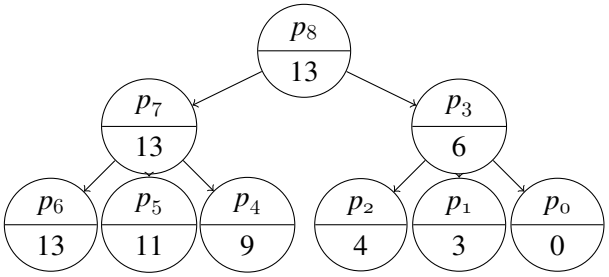
Example 5.4 (SHRINKING OF A SCHEDULE.) *Consider pages p_5 and p_6 from the schedule in Figure 5.6a. Pages p_5 and p_6 have visit-revisit intervals of length 9, although p_5 is hopeless and p_6 is hopeful. If we exchange their visit positions, p_6 gets visit-revisit interval of length 8 and page p_5 gets visit-revisit interval of length 10 as illustrated in Figure 5.6b.*

5.4.3. SHARC-Intervals Online Algorithm

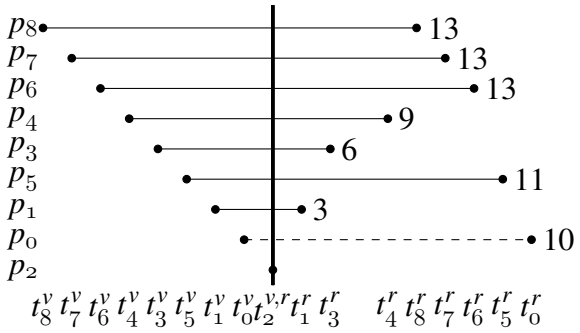
Here, we present an online version of SHARC-Intervals. Similarly to Section 4.4.4, we build a download schedule iteratively as we discover pages on the Web site. We still assume that the interval lengths can be predicted for each page, using a previously trained predictor based on page type and other features.

The offline version of SHARC-Intervals aims to place pages with short intervals close to the reference point. The online algorithm has a similar rationale, but has the handicap that it learns about interval lengths only as it detects new pages. This entails difficulties, since the online algorithm has to schedule the visits sequentially. For example, the online algorithm chooses the second visit, only after executing the first. In contrast, the offline algorithm first schedules all visits and afterwards executes them.

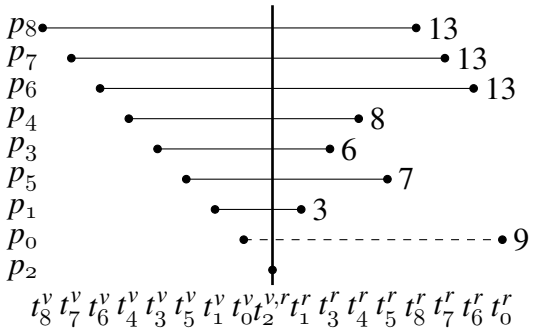
The online SHARC-Intervals starts with a set of seeds and aims to schedule the longest available interval. Selection of the longest interval gives as much room as possible for scheduling of shorter intervals around the reference point. If the longest interval is not possible to schedule, due to unavailable revisit position, we continue with the second



(a) Web Site



(b) Online Schedule before Shrinking



(c) Online Scheduling after Shrinking

Figure 5.7.: Online Example

longest interval. If no schedulable interval exists we schedule the shortest interval and declare it to be hopeless. The revisit positions of the hopeless pages are assigned at the end of all pages in reverse order. Once all pages are visited, we shrink the revisits. Individual steps of the algorithm are given in Algorithm 4.

Example 5.5 (ONLINE SCHEDULE.) *Consider the Web Site as in Figure 5.7a (here the lengths of the intervals are pointed out in the bottom part of the nodes) and the seed p_8. SHARC-Intervals online downloads the seed and discovers pages p_7 and p_3. p_7 has the longest interval, and therefore is scheduled and downloaded next. The list of detected pages increases to $\mathbb{I}^E = (p_6, p_5, p_4, p_3)$. Next p_7, and afterwards p_6 are the longest and schedulable pages, and therefore they are scheduled and visited. Now the next longest page is p_5 (of length 11), however it is not possible to schedule a revisit for the page, and p_3 is scheduled and visited instead. Pages p_5 and p_1 follow in a similar fashion. Since no page can be scheduled at this point, the shortest interval (page p_0) is visited and declared hopeless. Eventually, p_2 is visited and the visit part of the algorithm is completed. Now we shrink the revisits (the shortest intervals are shrunk having the highest priority). This way, pages p_4 (from length 9 to 8), p_5 (from length 11 to 7), and p_0 (from length 10 to 9)*

Algorithm 4 Online SHARC-Intervals

Require: Set of seeds $p_0^s, \ldots, p_m^s$
Number of pages in site n
Confidence threshold τ
Ensure: Downloaded sequence $(p_0^D, \ldots, p_n^D)$
Initialize the detected lengths sorted descending $\mathbb{I}^E = (I_0^s, \ldots, I_m^s)$
Initialize the revisits $\mathbb{R} = ()$
Initialize current visit position $v = -n$
while $\mathbb{I}^E \neq \emptyset$ **do**
  for all $I_i \in \mathbb{I}^E$ **do**
    Let $r = v + I_i$
    if $r > 0$ and $r \notin \mathbb{R}$ **then** ▷ //page p_i is hopeful
      Select page $p = p_i$ for download
      Exit the **foreach** loop
    end if
  end for
  if no page was downloaded in the **foreach** loop **then** ▷ //page p_j is hopeless
    Let $I_j \in \mathbb{I}^E$ be the shortest interval
    Select page $p = p_j$ for download
  end if
  Download p
  Update detected pages $\mathbb{I}^E = \mathbb{I}^E \cup \mathbb{I}^D(p_i)$
  if $r > 0$ and $r \notin \mathbb{R}$ **then**
    Use the revisit position $\mathbb{R} = \mathbb{R} \cup r$
  end if
end while
shrink($\mathbb{R}$)
Schedule revisits of hopeless pages in reverse order

get shorter intervals.

5.4.4. Estimation of the Threshold Parameter

Similarly to SHARC-Threshold, SHARC-Intervals is applicable in practice only after setting the threshold parameter τ. Using the same settings as in Section 5.3, we estimate the best value for τ. On three synthetic datasets with different change rate distribution, SHARC-Intervals achieves best values for $\tau = 0.90$ and $\tau = 0.95$ (cf. Table 5.2). Therefor, we select $\tau = 0.95$ for the experimental evaluation of SHARC-Intervals.

	SHARC-Intervals with varying values for τ									
Skew	$\tau = 0.1$	$\tau = 0.2$	$\tau = 0.3$	$\tau = 0.4$	$\tau = 0.5$	$\tau = 0.6$	$\tau = 0.7$	$\tau = 0.8$	$\tau = 0.9$	$\tau = 0.95$
1.00	8726	8574	8212	7831	7539	7358	7293	7273	7266	**7257**
1.50	1320	1291	1261	1212	1126	1047	970	874	**745**	790
2.00	240	231	221	211	195	176	165	130	104	**86**

Table 5.2.: SHARC-Intervals Incoherence with Varying τ

5.5. SHARC-Selective

Although SHARC-Threshold and SHARC-Intervals strategies improve the coherence of the captures, both strategies require a threshold parameter τ: either to determine the hopeless pages (SHARC-Threshold) or to compute the confidence intervals (SHARC-Intervals). The threshold parameter introduces an additional level of complexity and makes mathematical analysis of the strategies more difficult. In this section we introduce a the SHARC-Selective strategy which schedules visits and revisits without the need for a user defined parameter.

SHARC-Selective builds on the pyramid-like schedule of SHARC-Threshold (cf. Figure 5.2b). An alternative strategy without user defined parameters is to to try out all possible schedules of visit-revisit intervals ($\big((n+1)\big)^2$ in total) and opt for the strategy that has the highest *expected coherence*. The complexity of such a strategy makes it impractical. Focusing only on the family of visit-revisit schedules reduces the complexity of the task. Moreover, the expected coherence is higher for pyramid-like compared to equidistant schedule with equal change rates of the pages are the same. In addition, the mathematical basis of the SHARC-Selective strategy allows the formal definition of the hopeless pages.

In summary, SHARC-Selective employs three principles:

1. Visit-revisit intervals form a pyramid.

2. Greedily assign the hottest hopeful pages to the shortest intervals.

3. Greedily assign the hottest hopeless pages to the longest intervals.

We organize the section in the following. First, we define the expected coherence of a schedule. Second, we show that for the same change rates pyramid is better than equidistant schedule. Third, we define the hopeless pages, and give the SHARC-Selective offline and online algorithms.

Expected coherence of a schedule is the key concept to define hopeless page and SHARC-Selective schedule.

Definition 5.6 (EXPECTED COHERENCE OF A SCHEDULE.) *Let $\lambda_i, \ldots, \lambda_k$ be the change rates scheduled so the lengths of their visit-revisit intervals are $I_i, \ldots, I_k$. Then the expected coherence is*

$$EC\big((\lambda_i, I_i), (\lambda_{i+1}, I_{i+1}), \ldots, (\lambda_k, I_k)\big) \quad = \quad e^{-\lambda_i I_i} \quad + \quad \cdots \quad + \quad e^{-\lambda_k I_k}.$$

Example 5.6 (EXPECTED COHERENCE OF A SCHEDULE.) *Consider the schedule in Figure 5.8a. There the change rates* 0.40*,* 0.35*,* 0.30*,* 0.25*,* 0.20 *are scheduled on the intervals* $0, 2, 4, 6, 8$ *(hottest to shortest), and the expected coherence is*

$$EC\big((0.40, 0), (0.35, 2), (0.30, 4), (0.25, 6), (0.20, 8)\big) \\ = e^{-0.40 \cdot 0} + e^{-0.35 \cdot 2} + e^{-0.30 \cdot 4} + e^{-0.25 \cdot 6} + e^{-0.20 \cdot 8} \approx 2.22.$$

Therefore, expected coherence is 2.22 pages (out of maximum five coherent pages).

Pyramid is better than equidistant schedule in terms of expected coherence for the case when all pages have the same change rates. Below we formalize this result.

Theorem 5.2 (PYRAMID IS BETTER THAN EQUIDISTANT.) *Let $\lambda_0 = \cdots = \lambda_n = \lambda$. Then the expected coherence is higher for the pyramid schedule compared to the equidistant schedule for large enough n ($n \geq 1 + 1/(e^{\lambda\Delta} - 1)$).*

Proof of Theorem 5.2 *The proof is by induction. Assume that the theorem is true for the schedules of length n:*

$$\begin{aligned}
&\textit{expected-coherence}(\textit{pyramid}(n)) \\
&\quad = EC\big((\lambda,0)\ldots,(\lambda,2(n-1)\big) = 1+e^{-2\lambda\Delta}+e^{-4\lambda\Delta}+\ldots e^{-2(n-1)\lambda\Delta} \\
&\quad \geq 1+(n-1)e^{-n\lambda\Delta} = EC\big((\lambda,n)\ldots,(\lambda,n)\big) \\
&= \textit{expected-coherence}(\textit{equidist}(n)).
\end{aligned}$$

We need to prove that

$$\textit{expected-coherence(pyramid(}n+1\textit{))} \;\geq\; \textit{expected-coherence(equidist(}n+1\textit{))}.$$

Since

$$\begin{aligned}
&\textit{expected-coherence(pyramid(n+1))} = \\
&\qquad\qquad \textit{expected-coherence(pyramid(n))} + e^{-2n\lambda\Delta}
\end{aligned}$$

and

$$\begin{aligned}
&\textit{expected-coherence(equidist(n))} = \\
&\qquad \textit{expected-coherence(equidist(n-1))} + ne^{-(n+1)\lambda\Delta} - (n-1)e^{-n\lambda\Delta},
\end{aligned}$$

it suffices to show that

$$e^{-2n\lambda\Delta} \geq ne^{-(n+1)\lambda\Delta} - (n-1)e^{-n\lambda\Delta}.$$

This follows from the fact that

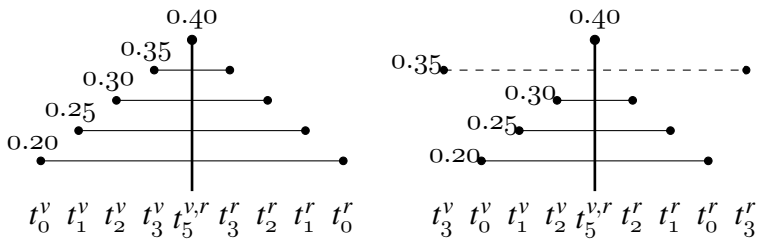
$$ne^{-(n+1)\lambda\Delta} \leq (n-1)e^{-n\lambda\Delta}. \tag{5.6}$$

Indeed, taking the logarithm of both sides in Equation (5.6):

$$\begin{aligned}
&\log(n) - (n+1)\lambda\Delta \leq \log(n-1) - n\lambda\Delta \Leftrightarrow \\
&\log(n) - \log(n-1) \leq \lambda\Delta,
\end{aligned}$$

which is true for all large enough n. □

A hopeless page is such an extremely hot page that it pays off to sacrifice the page and assign a long visit-revisit interval to it in order for the other pages to receive shorter intervals and increase the overall expected coherence of the capture.



(a) Hottest to Shortest (b) $\lambda_3 = 0.35$ is Hopeless

Figure 5.8.: Example with Hopeless Pages

Definition 5.7 (HOPELESS PAGE.) *Let $\lambda_i \leq \cdots \leq \lambda_k$ be the change rates and $I_i < \cdots < I_k$ be the lengths of the visit revisit intervals. Page p_i is hopeless (change rate λ_i is hopeless) if and only if*

$$\begin{aligned} &EC\big((\lambda_k, I_i), (\lambda_{k-1}, I_{i+1}), \ldots, (\lambda_i, I_k)\big) \\ &\quad = e^{-\lambda_k I_i} + e^{-\lambda_{k-1} I_{i+1}} + \cdots + e^{-\lambda_i I_k} \\ &\quad \leq e^{-\lambda_{k+1} I_i} + e^{-\lambda_{k+2} I_{i+1}} \cdots + e^{-\lambda_i I_{k-1}} + e^{-\lambda_k I_k} \\ &\quad = EC\big((\lambda_{k-1}, I_i), (\lambda_{k-2}, I_{i+1}), \ldots, (\lambda_i, I_{k-1}), (\lambda_k, I_k)\big). \end{aligned}$$

Otherwise we call page p_i (change rate λ_i) hopeful.

Example 5.7 (HOPELESS PAGE.) *Let $\lambda_1 = 0.20, \lambda_2 = 0.25, \lambda_3 = 0.30, \lambda_4 = 0.35$ and $I_1 = 2$, $I_2 = 4, I_3 = 6, I_4 = 8$ (cf. Example 5.6). Then change rate λ_1 is hopeless. In order to verify this statement we need to compare*

$$EC\big((0.35, 2), (0.30, 4), (0.25, 6), (0.20, 8)\big) \approx 1.22$$

(expected coherence for schedule in Figure 5.8a) with

$$EC\big((0.30, 2), (0.25, 4), (0.20, 6), (0.35, 8)\big) \approx 1.27$$

(the expected coherence for schedule in Figure 5.8b). Since $2.22 < 2.27$, therefore λ_3 is hopeless.

Algorithms. The SHARC-Selective (offline) employs the three principles to schedule the pages: (i) pyramid intervals, (ii) hottest hopeful pages to shortest interval, and (iii) hottest hopeless pages to the longest intervals.

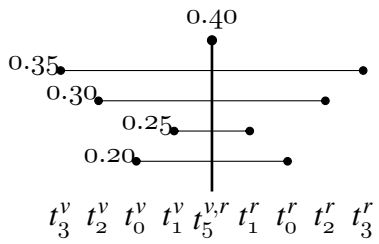


Figure 5.9.: SHARC-Selective Schedule for Change Rates in Figure 5.8a

Example 5.8 (SHARC-SELECTIVE.) *Let us continue Examples 5.6 and 5.7.*

Any page allocated the zero-length interval is always hopeful. Therefore $\lambda_5 = 0.4$ *is hopeful and allocated at the zero-length interval. This completes the first iteration.*

Example 5.6 shows that $\lambda_3 = 0.35$ *is a hopeless page for change rates* $(\lambda_3, \lambda_2, \lambda_1, \lambda_0) = (0.35, 0.30, 0.25, 0.20)$ *and intervals* $(I_1, I_2, I_3, I_4) = (2, 4, 6, 8)$ *and is allocated for interval* $I_4 = 8$*. This completes the second iteration.*

Since

$$EC\big((\lambda_2, I_1), (\lambda_1, I_2), (\lambda_0, I_3)\big) \approx 1.21 < 1.22 \approx EC\big((\lambda_1, I_1), (\lambda_0, I_2), (\lambda_3, I_3)\big) \quad (5.7)$$

therefore λ_2 *is also hopeless and is scheduled for the second largest interval* I_3*. This completes the third iteration.*

Since

$$EC\big((\lambda_1, I_1), (\lambda_0, I_2)\big) \approx 1.06 > 1.04 \approx EC\big((\lambda_2, I_1), (\lambda_1, I_2)\big), \quad (5.8)$$

λ_1 *is a hopeful change rate, and is allocated for the shortest interval* I_1*. This allocates the last (hopeful) change rate* λ_2 *to* I_2*. The schedule is illustrated in Figure 5.9.*

Algorithm 5 presents the SHARC-Selective offline algorithm. We check the page whether it is hopeless or hopeful, appending it to the corresponding queue. Once all hopeless pages $\mathbb{H}$ and hopeful pages $\mathbb{F}$ are identified, we visit all the hopeless pages $\mathbb{H}$ to allocate the the longest intervals to $\mathbb{H}$. Then hopeful page follow and get shorter intervals The revisits are in the reverse order.

SHARC-Selective offline schedules the pages from hottest to coldest around the middle point of the crawl. Consequently, the first Web page in the visit-revisit schedule is either the first detected hopeless page or the coldest page if there are no hopeless pages.

To turn the offline strategy into an online strategy, instead of scanning the pages from hottest to coldest, we scan from the coldest page to the hottest one. The strategy detects

Algorithm 5 SHARC-Selective Offline

Require: sorted pages $p_0, \ldots, p_n$
sorted intervals $I_0, \ldots, I_n$
Ensure: visit-revisit schedule $p_0^v, \ldots, p_n^v, p_0^r, \ldots, p_n^r$
hopeless pages $\mathbb{H}$ hopeful pages $\mathbb{F}$
Initialize hopeless pages $\mathbb{H} = ()$
Initialize hopeful pages $\mathbb{F} = ()$
for $i = 0, 1, \ldots, n$ **do**
  if $EC\big((\lambda_{n-i}, I_{|\mathbb{F}|}), \ldots, (\lambda_0, I_{n-|\mathbb{H}|})\big)\big) \geq EC\big((\lambda_{n-i-1}, I_{|\mathbb{F}|}), \ldots, (\lambda_{n-1}, I_{n-|\mathbb{H}|})\big)\big)$ **then**
    $\mathbb{F} = \mathbb{F} \cup p_i$
  else
    $\mathbb{H} = \mathbb{H} \cup p_i$
  end if
end for
visit $\mathbb{H}$; visit $\mathbb{F}$

on-the-fly new Web pages and schedules the visits of the hopeless pages as early as possible (so these pages get the longest available visit-revisit intervals). This is shown in Algorithm 6. We start with the set of seeds and identify hopeful and hopeless pages. If there are hopeless pages we download the hottest hopeless page so it gets the longest visit-revisit interval. Otherwise, we download the coldest hopeful page (we are at the bottom of the pyramid). Then we detect new pages, and again identify the hopeful and hopeless pages. The process is continued until all pages are downloaded. The algorithm concludes with downloads of all pages according to their scheduled revisits.

5.6. Experimental Evaluation

In this section, we present the evaluation of our visit-revisit strategies against a selected set of competitors. The section has a similar structure as the experimental section in Chapter **??** We introduce the competitors and give arguments for our choice of SHARC-Selective as a representative for the SHARC visit-revisit strategies in Section 5.6.1. In Section5.6.2 we briefly summarize the datasets on which we performed the experiments. The main experiments on coherence with real-world datastes are described in Section 5.6.3. We also execute live crawls with SHARC-Selective on a prototype which we developed.

Algorithm 6 SHARC-Selective Online

Require: sorted seeds $(p_0, \ldots, p_m)$
predicted Web size n
Ensure: visit-revisit sequence $(p_0^v, \ldots, p_n^v, p_0^r, \ldots, p_n^r)$
$(\mathbb{H}, \mathbb{F})$=sharcxi $(p_0, \ldots, p_m, 2m, \ldots, 2n)$
while $\mathbb{F} \neq \emptyset$ and $\mathbb{H} \neq \emptyset$ **do**
  if $\mathbb{H} \neq \emptyset$ **then**
    Download the hottest $p \in \mathbb{H}$
    $\mathbb{H} = \mathbb{H} \setminus p$
  else
    Download the coldest $p \in \mathbb{F}$
    $\mathbb{F} = \mathbb{F} \setminus p$
  end if
  $D = \text{urls}(p) \cup \mathbb{F} \cup \mathbb{H}$
  $(\mathbb{H}, \mathbb{F})$ = sharcxi$(D, 2|D|, \ldots, 2n)$
end while
revisit the pages in the opposite order

The results are presented in Section 5.6.4. Finally, in Section 5.6.5 we perform sensitivity studies where we systematically the properties of the synthetic datastes like size, change skew, and crawl duration. As in Chapter 4, the experiments were designed so that we could use the exact history of changes as a reference for computing the coherence of the captures.

5.6.1. Methods under Comparison

We evaluate the online version of SHARC-Selective against the offline variant which we use as a baseline and against the competitor strategies we introduced in Chapter 4: Breadth-first search (BFS), Depth-first search (DFS), Hottest-first (HF), and Hottest-last (HL). Since we evaluate visit-revisit strategies we extend the baseline strategies with revisits. We include two version of the revisits - FIFO (first-in-first-out) or LIFO (last-in-first-out). In practice, this means that the FIFO strategies schedule the revisits the same way as the revisits, while the LIFO strategies schedule the revisits in reverse order. BFS and DFS schedule the revisits based on the graph structure, while Hottest-first and Hottest-last schedule the revisits based on the change rates. Thus, we consider Hottest-last LIFO

as baseline of the competitor strategies, since its schedule is similar to SHARC-Selective but without the hopeless pages.

We prefered SHARC-Selective to SHARC-Threshold and SHARC-Intervals for the evaluation since an earlier investigation on a older version of the MPII dataset (cf. Table 5.3) showed that SHARC-Threshold and SHARC-Intervals configured with the best value for the threshold parameter τ performed at most as good as Hottest-last LIFO. Moreover, SHARC-Threshold consistently outperformed SHARC-Intervals. Table 5.4 contains the results of the investigation. These results led to the design of SHARC-Selective which is based on the pyramid-like schedule of SHARC-Threshold but without the need of a threshold parameter τ. SHARC-Selective automatically adjusts its schedule such that the expected coherence is maximized.

Dataset	Web Site	Periodicity	Pages	Changing Pages
MPII	mpi-inf.mpg.de	daily	72394	687

Table 5.3.: Earlier Version of the MPII Dataset

Thresholds	$\tau = 0.1$	$\tau = 0.2$	$\tau = 0.3$	$\tau = 0.4$	$\tau = 0.5$	$\tau = 0.6$	$\tau = 0.7$	$\tau = 0.8$	$\tau = 0.9$
Hottest-Last LIFO	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**
SHARC-Intervals	221	214	214	215	215	71	71	**3**	54
SHARC-Threshold	172	155	136	114	96	79	57	28	**3**

Table 5.4.: Incoherence with Oracle with Varying τ on MPII

5.6.2. Datasets

We tested our methods on the real-world datasets introduced in Chapter 3, on the live Web sites with sitemaps summarized in Table 5.5, and also on synthetically generated Web sites for systematic variation of site properties.

The real-world datasets are MPII (an academic Web site), DMOZ (a Web directory), and MOD, DFID, ARMY, RAF, and DH (all of them governmental Web sites). The size varies from around 2 000 Web pages (DH) to over 177 000 (DMOZ). We use the datasets in the main experiments on coherence (cf. Section 5.6.3). The experiments are performed with artificially slowed down crawls that cover the whole time periods of the datasets. Experiments with live crawls were executed as well (cf. Section 5.6.4).

Dataset	Web Site	Pages
DeutscheWelle	dw-world.de	5309
Intereconomia	intereconomia.com	2948
EuroNews	fr.euronews.net	672

Table 5.5.: Live Web Sites with Sitemaps

For the sensitivity studies we used the same synthetic datasets as in Chapter 4. We simulate changes according to the Poisson process. The change rates are assigned according to a skewed distribution: $\lambda_i = 1/((i+1)^{\text{skew}})$. The Web graph is a tree with n pages. The outdegree of each page p_i is constant outdegree. Here are the default values of the parameters: skew $= 1.2$, outdegree $= 400$, and $n = 10,000$. We keep both "Cold Leaves" and "Hot Leaves" flavours of the tree. The "Cold Leaves" tree has the hottest page as root and then "cools down", leaving the coldest pages on the leaves. The "Hot Leaves" tree follows the reverse pattern: coldest page is the root of the tree and pages "heat up" towards the leaves.

5.6.3. Coherence Experiments with with Real-World Datasets

Similarly to Chapter 4, we estimate change rate estimates with either an oracle or a predictor. In the experiments we measure and report the *actual incoherence*: the number of pages that changed during their visit-revisit intervals. So just like with blur, the lower the reported numbers are the better it is for Web archive quality.

We simulated artificially slowed down crawls on the datasets so that they would take as long as the entire time period covered by the dataset. Although not realistic, this setting helps get more informative stress-test results.

The results of the experiments on incoherence with visit-revisit strategies over real-world datasets are shown in Table 5.6 with change rate oracle and Table 5.7 with change rate predictors, respectively.

For the tests with change rate oracle, the SHARC-Selective strategies outperformed all baseline opponents by a substantial margin. SHARC-Selective offline exhibits incoherence values that are lower than those of the competitors by more than a factor of 3. SHARC-Selective online did not perform quite as well as its offline counterpart, but it is not much worse and still much better than all online competitors. On one specific dataset, SHARC-Selective online outperformed the offline variant, but it is due to "random" effects

Site	SHARC-Selective Offline	SHARC-Selective Online	SHARC Revisits	HF FIFO	HF LIFO	HL FIFO	HL LIFO	BFS FIFO	BFS LIFO	DFS FIFO	DFS LIFO
MPII	**145**	453	837	1013	1252	751	508	935	1091	908	726
DMOZ	**15914**	25206	34233	37405	42240	33413	31720	35672	36176	34357	33048
MOD	**4449**	4701	5803	5903	5769	5766	5452	5840	5475	5767	4890
DFID	**959**	987	1113	1110	1115	1109	1104	1097	1038	1079	1049
ARMY	**10947**	12479	14052	13865	14528	14008	13901	14104	13996	14172	13120
RAF	**2120**	3284	4096	3487	4243	4077	3676	3242	3192	4006	3941
DH	11376	**11368**	11589	11514	11494	11546	11457	11625	11483	11625	11482

Table 5.6.: Incoherence with Oracle of Change Rates

Site	SHARC-Selective Offline	SHARC-Selective Online	SHARC Revisits	HF FIFO	HF LIFO	HL FIFO	HL LIFO
MPII	558	**504**	720	835	931	808	**504**
DMOZ	**26016**	31847	34098	36553	38396	34447	31847
MOD	**4516**	5089	5776	5867	5662	5748	5262
DFID	**961**	985	1076	1102	1076	1088	1054
ARMY	**11774**	12876	14305	14290	14245	14113	13796
RAF	**3520**	3745	3891	4030	3615	3914	3745
DH	11360	**11305**	11545	11596	11517	11681	11612

Table 5.7.: Incoherence with Predictor

regarding lucky situations by the order in which pages are detected. SHARC-Revisits is designed to minimize the blur metric (not shown here), and performed moderately on incoherence. The flexibility that the SHARC-Selective strategies have in dealing with hopeless pages pays off well and leads to the highest gains on sites with a large number of changing pages like DMOZ, MOD, DH, and ARMY.

When all methods are limited to the realistic case of relying on a change rate predictor, the SHARC-Selective strategies again win by a substantial margin, as shown in Table 5.7. We did no longer include the BFS and DFS strategies in this comparison, as they were already clear losers in the experiments with change rate oracle. For some datasets, SHARC-Selective online even performed better than the offline variant. The explanation lies in the nature of the corresponding Web sites. For example, the MPII dataset is very skewed: there are about 1,000 pages, out of 70,000, with very high change rates, while the other 69,000 hardly ever changed. This reduced the quality of the change rate predictor

and led to suboptimal behavior. SHARC-Selective online, our natural candidate for deployment in a real system, performed very well across the suite of datasets: usually not much worse than the offline variant, and sometimes even better.

Compared to the previous experiment with change rate oracle (see Table 5.6), the incoherence values of the predictor-based methods increased considerably. Although our predictors generally provided decent accuracy, there is room for improvement in this regard.

5.6.4. Live Experiments with Sitemaps

We also performed live measurements (as opposed to experiments replayed from stored data about former crawls) on a number of sites, using our prototype based on the Heritrix crawler. Here we focused on sites which publish sitemaps, and we limit ourselves to the three most interesting competitors because each strategy requires a separate crawl in real-time. Running more simultaneous crawls on the same Web site would influence the crawls themselves, and running different strategies sequentially would make their results incomparable as they no longer see the same state of the site.

We crawled selected subsites of `dw-world.de` (5309 pages in total), `intereconomia.com` (2948 pages), and `fr.euronews.net` (672 pages) defined by available sitemaps. The results are shown in Table 5.8. SHARC-Selective clearly wins for all sites.

Site	SHARC Selective	HL LIFO	DFS LIFO
DeutcheWelle	**3655**	3705	3712
Intereconomia	**2704**	2723	2721
EuroNews	**643**	657	658

Table 5.8.: Incoherence with Sitemaps and Heritrix Crawler

5.6.5. Sensitivity Studies

Similarly to the evaluation of the blur-minimizing strategies, we studied the effects on SHARC-Selective from the scale (size) of a Web site, the skew in the change rate distribution of a site's pages, and the politeness-driven duration of the crawl.

Scalability. In this experiment we vary the size of the Web site. Tables 5.9 and 5.10 show results for the incoherence measure with visit-revisit strategies. SHARC-Selective strategies outperform all competitors, with increasing gains as the size of the site increased. SHARC-Selective online produces schedules identical to those of the offline variant for hot pages near the root (Table 5.9), as it detects these pages early. For the dual case of hot pages near the leaves (Table 5.10), SHARC-Selective online lost against its offline counterpart, but still outperformed all other opponents by a large margin.

	SHARC-Selective		SHARC	HF	HF	HL	HL	BFS	BFS	DFS	DFS
Size	Offline	Online	Revisits	FIFO	LIFO	FIFO	LIFO	FIFO	LIFO	FIFO	LIFO
$1 \cdot 10^4$	**3075**	**3075**	4104	4184	4140	4177	3079	4170	3719	4163	3995
$5 \cdot 10^4$	**11940**	**11940**	17327	17394	17959	17397	12100	17368	16072	17418	16366
$1 \cdot 10^5$	**21378**	**21378**	32051	32142	33705	32110	21533	32015	29033	32129	29395

Table 5.9.: Scalability: Incoherence (Cold Leaves)

	SHARC-Selective		SHARC	HF	HF	HL	HL	BFS	BFS	DFS	DFS
Size	Offline	Online	Revisits	FIFO	LIFO	FIFO	LIFO	FIFO	LIFO	FIFO	LIFO
$1 \cdot 10^4$	**3019**	3039	4143	4140	4057	4165	3238	4136	3785	4175	3999
$5 \cdot 10^4$	**11830**	11997	17377	17321	17925	17330	12364	17288	16582	17366	15397
$1 \cdot 10^5$	**21417**	21630	32295	32050	33776	32313	22165	32028	30618	32181	29664

Table 5.10.: Scalability: Incoherence (Hot Leaves)

Crawl Duration. To increase the crawl duration we multiply the politeness delay by a slow-down factor. For visit-revisit strategies aiming at low incoherence, the results are shown in Tables 5.11 and 5.12. The SHARC-Selective methods outperform all competitors. The online method is very close to the offline variant, and sometimes even better (see Table 5.11). This is due to "random" effects: early discovery of hopeless (very hot) pages resulted in almost identical schedules for online and offline methods. But for the dual case with hot pages closer to leaves (Table 5.12), the offline strategy consistently outperforms the online variant as the latter discovers hopeless pages much later and thus places them in suboptimal slots.

Skew. The skew parameter determines the distribution of changes are among the pages. If the skew is high most of the changes go to very few pages. Conversely, low skew

Slow-down	SHARC-Selective Offline	SHARC-Selective Online	SHARC Revisits	HF FIFO	HF LIFO	HL FIFO	HL LIFO	BFS FIFO	BFS LIFO	DFS FIFO	DFS LIFO
1	**3058**	**3058**	4215	4172	4118	4212	3062	4176	3931	4222	3695
2	4608	**4607**	5865	5875	5359	5857	4919	5811	5317	5887	5542
3	**5672**	5673	6945	7009	6193	6984	6301	6984	6215	6975	6336
4	**6255**	6286	7695	7678	6646	7711	7287	7701	6814	7703	6842
5	**6721**	6753	8207	8214	7009	8224	7985	8224	7221	8195	7414
10	7906	**7903**	9443	9419	8003	9432	9488	9415	8561	9425	8619

Table 5.11.: Crawl Duration: Incoherence (Cold Leaves)

Slow-down	SHARC-Selective Offline	SHARC-Selective Online	SHARC Revisits	HF FIFO	HF LIFO	HL FIFO	HL LIFO	BFS FIFO	BFS LIFO	DFS FIFO	DFS LIFO
1	**3002**	3006	4102	4115	4083	4150	3216	4091	3735	4170	4006
2	**4547**	4859	5778	5767	5281	5741	5288	5733	5598	5818	5305
3	**5552**	6387	6835	6894	6066	6833	6774	6896	6531	6887	6260
4	**6221**	6466	7697	7689	6696	7688	7816	7690	6856	7681	6953
5	**6744**	6847	8208	8215	7092	8210	8541	8222	7379	8195	7634
10	**7959**	8075	9431	9436	8118	9414	9760	9403	8708	9446	8309

Table 5.12.: Crawl Duration: Incoherence (Hot Leaves)

distributes the changes uniformly. Tables 5.13 and 5.14 show the result of the study with visit-revisit strategies aiming at low incoherence. SHARC-Selective offline is the best strategy and SHARC-Selective online is second best. For large skew values (skew > 1) and hot pages closer to leaves, the online method does not differ from the offline variant, since very few pages are very hot and their late discovery by the online crawler does not influence the schedule anymore.

Skew	SHARC-Selective Offline	SHARC-Selective Online	SHARC Revisits	HF FIFO	HF LIFO	HL FIFO	HL LIFO	BFS FIFO	BFS LIFO	DFS FIFO	DFS LIFO
1.00	**7061**	7073	8518	8517	7257	8504	8110	8540	7623	8509	7567
1.50	**365**	367	1064	1037	1308	1060	710	1056	1119	1035	935
2.00	**11**	35	169	174	246	176	210	178	223	170	175

Table 5.13.: Skew: Incoherence (Cold Leaves)

	SHARC-Selective		SHARC	HF	HF	HL	HL	BFS	BFS	DFS	DFS
Skew	Offline	Online	Revisits	FIFO	LIFO	FIFO	LIFO	FIFO	LIFO	FIFO	LIFO
1.00	**7012**	7151	8455	8479	7283	8475	8587	8516	7790	8486	7316
1.50	**355**	**355**	1018	1010	1237	1033	362	1035	714	1022	1157
2.00	**20**	**20**	182	174	225	171	22	177	186	181	169

Table 5.14.: Skew: Incoherence (Hot Leaves)

5.7. Summary

In this chapter we defined the coherence quality measure for Web archives. It is a deterministic measure appropriate for legal use of Web archives. For deterministic guarantees we introduced crawl strategies that visit pages twice: a first visit to fetch the page and a later revisit to validate that the page has not changed. We developed four distinctive strategies: SHARC-Revisits, SHARC-Threshold, SHARC-Intervals, and SHARC-Selective. SHARC-Revisits simultaneously gives deterministic guarantees for the capture and minimizes the capture's blur. The other three strategies aim at maximizing the coherence. The main difference is that SHARC-Selective does not need user defined parameters to schedule the pages for download. The only requirement for good performance of SHARC-Selective is an adequate estimation of the change rates of the Web pages. The next chapter covers the change prediction mechanisms that we employ.

Chapter 6.

Prediction of Changes

In this thesis, we assume that each Web page changes according to a Poisson process with an average change rate λ. This model is a simple, yet powerful tool to measure and improve Web archive quality. The concepts of blur and coherence introduced in Chapter 3 are mathematically expressed using the properties of the Poisson process. The model is also the basis of the algorithms for download schedules yielding high-quality Web archives. The insights from the analysis of the algorithms (organ-pipe schedule have lowest blur and hopeless pages may improve coherence) are applicable in the design of algorithms based on other change prediction models like periodicity and burstiness.

Decent change rate estimation is the crucial requirement for the application of the strategies proposed in Chapters 4 and 5. Change rates can be determined from three sources: 1) extracted from sitemaps, 2) estimated from previous crawls of a site, 3) predicted by machine learning methods (classifiers or regression models) based on easily observable features. We discuss all these issues below in turn.

6.1. Sitemaps

Sitemaps are an easy way for webmasters to inform robots about pages on their sites that are available at the Web site for crawling. Recalling the background information provided in Chapter 2, sitemaps are XML files that contain URLs pointing to other sitemaps or a list of URLs available at the site. The compressed size of the sitemap is limited to 10MB and can contain up to 50K URLs. These limitations are introduced so that the Web server does not need to serve very large files. If a sitemap exceeds the limit, then multiple sitemap files and a sitemap index file must be created. However, it has become practice that webmasters create several sitemaps even for small Web sites, grouping the URLs into

conceptual partitions of interrelated URLs on a site, *sub-sites* so to speak. Our framework can harness information about sub-sites that site owners want to be crawled and archived as coherently as possible.

A sitemap file consists of a list of URLs with metadata indicating the time of last modification, the change frequency, and the priority of the URLs. The corresponding fields in the sitemap file are **lastmod**, **changefreq**, and **priority**. The *changefreq* field may have the following values *always, hourly, daily, weekly, monthly, yearly, never*. In our implementation, however, we work with numerical values instead of categorical ones. We map every category to a constant which describes the change behaviour of the pages in the sitemap.

If our time unit is 1 minute, then a straightforward mapping would map 1 to the *always* category, $1/60$ to the *hourly* category and so on. Such a mapping implies that pages on average change once in a time interval of a certain length (hour, day, week, month, year). This is very restrictive, since a page may change twice a day or four times a week. We propose a mapping scheme which takes into consideration that the change rate categories in the sitemaps are only rough estimates and the actual change rates of the Web pages may vary.

The mapping scheme defines a range of change rates for a category with an associated time interval of a certain length. Then the average change rate in the range is taken as the change rate that corresponds to the category. The change rates are such that the probabilty that a page changes in the associated time interval is greater than a threshold parameter. The formal definition follows:

Definition 6.1 (CHANGE RATES OF SITEMAP CATEGORIES) *Let $C_0, C_1, \ldots, C_n$ be change rate categories. Let $C_0, C_1, \ldots, C_{n-1}$ be associated with time intervals of length $T_0, T_1, \ldots T_{n-1}$ such that $T_i < T_{i+1}$ for $0 \leq i < n$. Let τ be a threshold parameter. Then the change rates $\lambda_0, \lambda_1, \ldots \lambda_n$ corresponding to $C_0, C_1, \ldots, C_n$ are defined as*

$$\lambda_i = \begin{cases} 1 & \text{for } i = 0, \\ \frac{T_{i-1}+T_i}{2T_{i-1}T_i} \log \tau^{-1} & \text{for } 0 < i < n, \\ \frac{1}{2T_{i-1}} \log \tau^{-1} & \text{for } i = n. \end{cases}$$

In order to assign change rate to every change rate category of the Sitemap protocol, we associate the *always* category with a time interval with the length of the politeness delay Δ. We set $\tau = 0.95$, since it gives us strong confidence that a page changes at least once in a time interval indicated by its change rate category.

Although sitemaps become increasingly popular and publish information about the change frequency of the Web pages, estimating change rates from previous crawls is still prefered.

6.2. Estimation of Change Rates from Previous Crawls

We estimate change rates of Web pages with the so called *oracle of change rates*. We call it an oracle, since it needs to know the full history of changes. This is in contrast to a change rate *predictor* (see the next section) where the change history is known only for a *sample* of pages and is used to learn a prediction model.

We use the standard maximum likelihood estimator (MLE) for a Poisson distribution to estimate the change rate λ. Given a smaple of n observations $x_1, \ldots, x_n$ the MLE is defined as

$$\hat{\lambda} = \frac{1}{n} \sum_{i=1}^{n} x_i.$$

We postulate that the history of timepoints of changes of a page p for a time period T is available. Given a politenes delay Δ such that $T = n\Delta$, we split the time period T into n equal intervals of length Δ. The observation x_i represents the number of changes observed in the i-th interval. Then the estimation of the change rate of p is given by the MLE, under the assumption that the changes of p follow a Poisson process.

In practice not all Web pages may have history of changes. In the next section, we demonstrate that it is possible to predict the change rates of a newly discovered Web page based on features extracted from the URL and the content of the Web page.

6.3. Prediction of Change Rates with Classifiers

The predictor of change rates uses Naive Bayes and the C4.5 decision-tree classifiers to predict change rates from given features of a page. We also tried linear regression, but the due to the large number of non-changing pages the predicted change rates were overwhelmingly close to zero. These initial results and the choice of categorical features, especially in the online case, influenced our decision to use classifiers.

Since classifiers work with categorical output data (labeled classes), we discretize the change rates using equal-frequency binning [46] with ten bins. Equal-frequency binning

aims to partition the domain of change rates into bins (intervals) so that each bin contains the same (or nearly the same) number of observations (individual pages) from the dataset. As for the features of the pages, we have investigated two different sets: features that are only available in online settings (the Web page itself is not available, but only its URL and its metadata) and offline settings (where the Web page is available as well):

- **online features:** features from the URL string:
  - the domain name
  - the MIME type – the type of the Web pages, it is available in the HTTP response from the server,
  - the depth of the URL path – the number of slashes in the URL,
  - the number of characters in the URL,
  - the first three word-segments of the URL path, a word-segement in a URL is the substring between two slashes,
  - boolean flags indicating presence or absence of special symbols in the URL: tilde(~), underline(_), question mark(?), semi-column(;), column(:), comma(,), and percentage sign (%) .
- **offline features:** all online features and the following additional features:
  - the number of day since the last change,
  - the number of images in the Web page,
  - the number of tables in the Web page,
  - the number of outlinks from the Web page,
  - the number of inlinks to the Web page.

6.4. Evaluation

We tested the classifiers on the available Web archive and the sitemap datasets.

Although we already introduced the datasets in earlier chapters, for the purpose of clarity we describe them again. The Web archive datasets are periodic Web archives of an academic Web site (MPII), a Web directory (DMOZ), and governmental Web sites (MOD, DFID, ARMY, RAF, DH). The sitemap datasets (DeutscheWelle, Intercomnia, EuroNews)

are based on the sitemaps published by three news agencies. Tables 6.1 and 6.2 summarize the size of the datasets.

Dataset	Web Site	Pages	Changing Pages
MPII	mpi-inf.mpg.de	72,071	1,356
DMOZ	dmoz.org	177,446	50,855
MOD	mod.uk	10,047	5,988
DFID	dfid.gov.uk	2,186	1,131
ARMY	army.mod.uk	37,330	15,259
RAF	raf.mod.uk	27,836	4,286
DH	dh.gov.uk	15,884	12,203

Table 6.1.: Web Aarchive Datasets

Dataset	Web Site	Pages
DeutscheWelle	dw-world.de	5309
Intercomnia	intereconomia.com	2948
EuroNews	fr.euronews.net	672

Table 6.2.: Sitemap Datsets

The methodology we use for evaluation is 10-fold crossvalidation. We split every dataset into 10 folds. We select 9 folds for training and one for testing. We repeat this procedure 10 times, such that every fold serves as test data. The reported precision is the average value of all 10 test iterations.

The results are shown in Table 6.3.

The change rate predictors are indeed practically viable. Not surprisingly, the overall winners use offline features, but the online-features predictors are also fairly accurate.

C4.5 is slightly more accurate than the Naive Bayesian classifier (see Table 6.3); therefore we use C4.5 in our main experiments presented in Section 4.5. For most datasets the classifier achieved about 70% accuracy. However, even when the classifier is only 40% accurate, the mispredicted change rates are typically close to the actual values and still useful for the scheduling algorithms. This is because we are using ten bins and when we do not classify the change rate into the correct bin (for example the bin with

Dataset	Online Features		Offline Features	
	Bayesian	C4.5	Bayesian	C4.5
MPII	90.857	**97.951**	97.502	**97.951**
DMOZ	27.398	**43.520**	48.670	**43.753**
MOD	79.379	**85.995**	80.794	**86.886**
DFID	**70.653**	67.437	**74.020**	71.256
ARMY	78.423	**80.660**	79.312	**82.406**
RAF	85.848	**86.561**	89.453	**91.902**
DH	37.011	**41.658**	41.798	**46.002**

Table 6.3.: Classification Precision for Web Archive Datasets

hourly change rates) it is often assigned to an adjacent bin (e.g., the bin of daily changing rates).

We also tested the classifiers for sites with the sitemap datasets. The setup of the experiment is similar to the previous setting; however, we did not need to discretize the change rates, because change frequency is already a categorical attribute in sitemaps (see sitemap example at the beginning of the section). As input features of pages, we have used the domain name, MIME type, depth within the URL path, and number of days since the last update.

As in the previous experimetn, we applied 10-fold crossvalidation. The datasets were split into 10 folds, 9 were used for training and one for testing. We took the averge precision of the 10 different test runs. The results of cross-validation are shown in Figure 6.4. Change rate predictors for sitemaps are extremely precise.

Dataset	Naive Bayesian	C4.5
DeutscheWelle	**100.00**	99.981
Intereconomia	98.464	**99.889**
EuroNews	**99.977**	**99.977**

Table 6.4.: Classification Precision for Sitemap Datasets

Chapter 7.

Prototype Implementation

We designed and implemented a prototype based on the Heritrix Web crawler [65]. The source code of the prototype is available online [1] as a module in the Living Web Archives project. In this chapter, we describe the prototype's architecture, algorithms, and data structures.

7.1. Prototype Architecture

The Heritrix Web crawler is a Web crawler designed especially for the task of Web archiving. It is the crawler of choice in various archiving institutions and high-profile libraries (Bibliothèque nationale de France[2], CiteSeerX[3], Smithsonian Institution Archives[4]). Three factors have contributed most for its wide adoption. First, it is an open-source project. developed by Internet Archive with contributions and support from other institutions. Second, it is addresses the special needs of Web archives. In a joint effort, twelve national libraries together with the Internet Archive collected a list of detailed requirements for an Web archiving crawler [44]. Heritrix has been developed with these requirements in mind. Finally, Heritrix's architecture allows for easy customization.

Figure 7.1 shows the architecture of Heritrix. The basic units of the crawler are the crawl jobs, the crawl controller and the configuration interfaces.

Users interact with Heritrix by providing crawl configurations. Users can do that either through a Web-based interface or through an application programming interface.

[1] http://code.google.com/p/liwa-technologies/

[2] http://www.bnf.fr/

[3] http://citeseerx.ist.psu.edu/

[4] http://siarchives.si.edu/

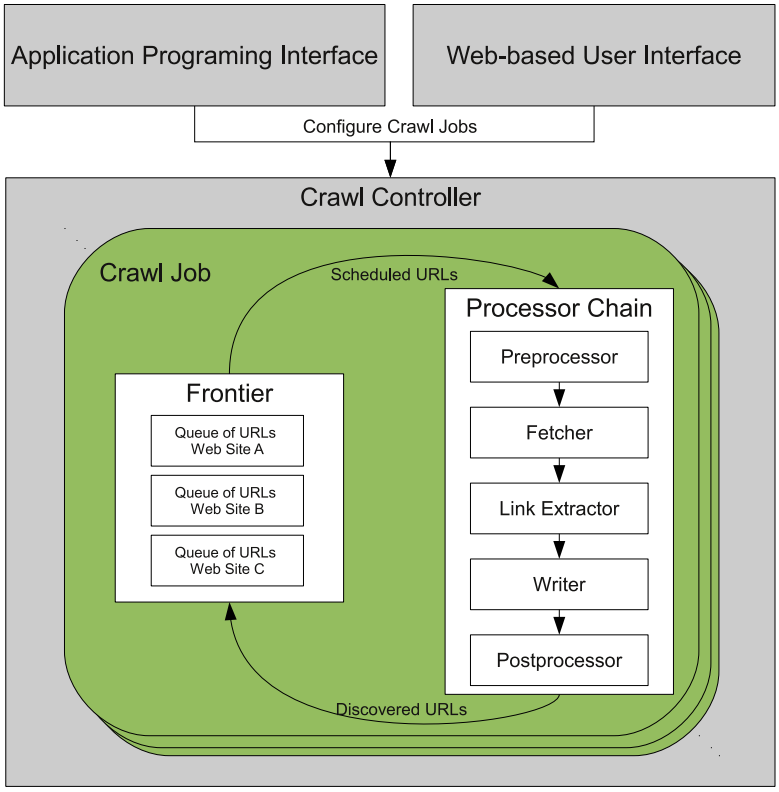


Figure 7.1.: Heritrix Architecture

Providing these two options of interaction, the Heritrix Web crawler can be used both as a standalone application or as a separate module in a bigger system.

The configuration files include parameters of a crawl such as seeds, scope, size, duration, politeness delay, broadband restrictions as well as definitions of data structures and other modules like database connectivity, download strategies, and protocol support. For each configuration file, the crawl controller creates, starts, and monitors a crawl jobs.

A **crawl job** manages the execution of a crawl. It consists of two main modules: a processor chains and a frontier.

A processor chain is a sequence of objects (processors) which handle a detected URL in a fixed order. Heritrix comes with a default processor chain. If there is a need, one can modify the processor chain by adding or removing processors. The default processor chain consists of preprocessor, fetcher, link extractor, writer, and postprocessor. The preprocess determines if a URL belongs to the scope of the crawl. The scope definition may contain restrictions on various properties of the URL (domain, MIME type, hierarchy

depth, etc.). If the URL is selected for download, it is passed to the fetcher. The fetcher selects the appropriate protocol for download (HTTP, FTP, DNS, etc.) and downloads the URL. The content of the URL is passed to the link extractor which parses the content and extracts outlinks. The link extractor finds outlinks not only in HTML files, but also in JavaScript, CSS or PDF files. The URLs of the outlinks are stored in the frontier. After extracting the outlinks, the writer saves the page content in ARC [31] or WARC[32] files. Finally, the postprocessor updates system variables in Heritrix signaling that the processor chain is ready to handle a new URL.

All detected and not yet downloaded URLs are stored in the frontier. The frontier maintains different queues for URLs from each Web site. This is done to achieve high throughput without overloading the remote servers. When the process chain is ready for a new URL, the frontier picks up a URL from the queue corresponding to the least queried Web server and feeds the process chain. However, the order of the URLs in the queue is fixed. This limitation of Heritrix does not allow embedding a scheduling algorithm inside the frontier. Instead, the scheduling takes place in the crawl controller. Figure 7.2 shows the architecture of our SHARC prototype and its relation to Heritrix.

The prototype has two main modules: scheduler and database. The **scheduler** dispatches pages for downloading, driven by configurable options for the selected download strategy. In the original Heritrix crawler, the scheduling is based on a breadth-first strategy; search engines, on the other hand, employ techniques that optimize for freshness, importance of pages, and scope (news, blogs, Deep Web).

In this prototype, we implement visit-revisit schedules. It allows testing for content changes right after the crawl has completed. Several visit-revisit strategies are supported including SHARC-Selective. The scheduler creates a download schedule for visits and revisits of URLs according to the selected strategy. The URLs are inserted in the queues of the frontier in the prescribed order. Each URL is passed to the processor chain and subsequently crawled. To make the crawling more efficient, we configure Heritrix to use conditional GET requests that make use of the contents' ETags. As a result, the revisit phase becomes faster by simultaneously reducing bandwidth as well as server load. All crawls are then made accessible as a set of distinct visit-revisit pairs. The visits and the revisits are stored and indexed in a **database** (PostgreSQL in our case).

While we reused most of the modules of Heritrix and we also added three more modules to Heritrix (the shaded region in Figure 7.2).The **sitemaps** and the **seeds** modules provide a list of URLs for the crawl. The seeds module loads the URLs from exisiting Web archives. The sitemap module gets the URLs from a set of specified sitemaps.

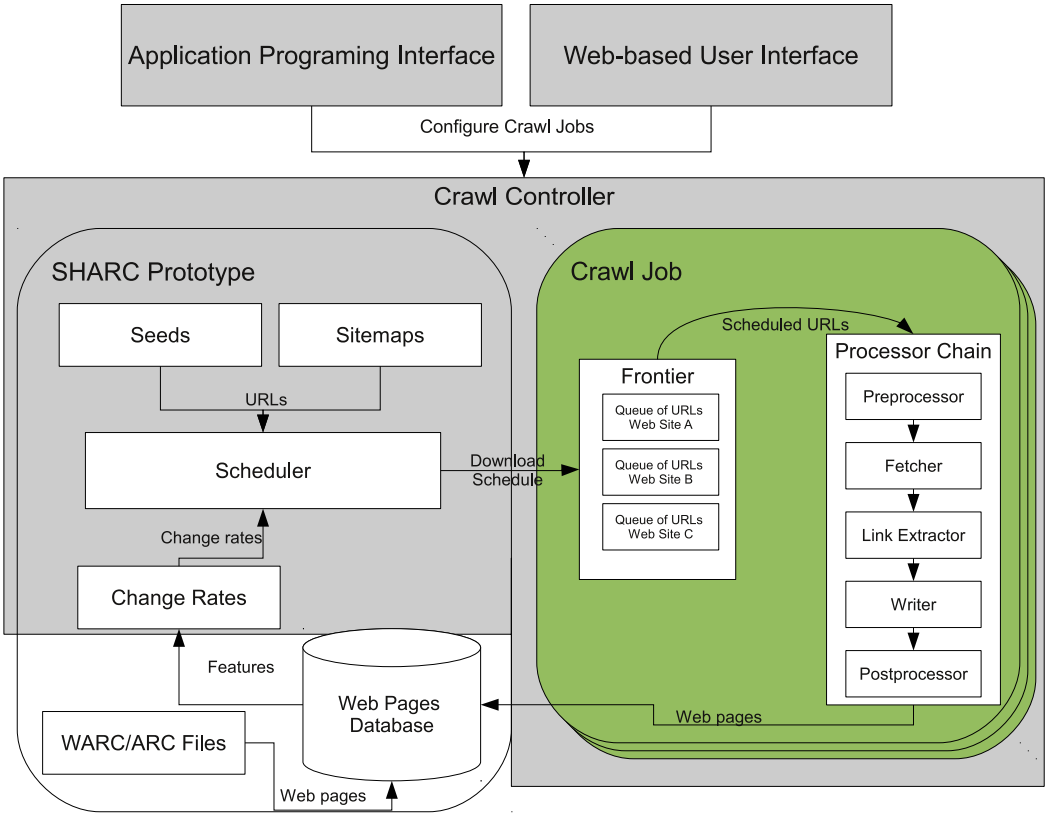


Figure 7.2.: SHARC Architecture

Additionally, this module supplies the change frequencies of the Web pages as given in the sitemaps. In case no sitemaps are used, the scheduler receives the change rates from the **change rate** module.

The change rate module either estimates change rates from previous crawls of a site or predicts them by machine learning techniques. Chapter 6 gives more details on these methods. Both the statistic for the change rate estimation and the training data for machine learning algorithms come from our database of Web archives. Apart from storing archives from Heritrix, we import ARC and WARC files to our database for better change rate prediction and change analysis.

7.2. Data Extraction and Preparation

This section discusses data collection, extraction, and preparation issues for the understanding and prediction of changes in Web archives.

Different types of change analysis and prediction techniques may pose different requirements for the input data. In the simplest case an analysis may require a single instance of a site's archived pages, while classifiers and more elaborated analysis may investigate the dynamics of changes requiring a sequence of captures of one or multiple sites including both page (content) and link (structure) information of the site(s). To cope with the generality of the inputs we aim to reuse as much of the general database management technologies as possible and push as much of data storage, retrieval, and processing to the standard SQL DBMS level. In this section we outline how a database schema should look like, and how to import and clean the data with standard SQL.

Database Schema

Essentially, the database schema (cf. Figure 7.3) consists of the pages relation (t_pages) and links relation (t_links). The pages relation records information related to a page in a Web site including its url, size, status code, and last modified timestamp. In addition, we encode the url with url_id but record only the site_id. This allows us to quickly and efficiently retrieve selections of the pages of specific site_id and crawl_id, check whether the page has changed in two subsequent crawls, and perform efficient and effective data cleaning (cf. Section 7.2). The payload of the page is stored in the content attribute provided it did change compared to the previous crawl (cf. vs_page_id). Essentially, the link information is recorded in t_links table. The pair attributes from_url_id to_url_id identify all links from and to a page for a given crawl. The crawl order of the Web archive can be accessed through the parent_page_id attribute in the t_pages table.

Data Import from WARC files

Data import from ARC and WARC files primarily consists of two tasks: loading the data into the database, and reduplicating and cleaning the data.

ARC [31] and its successor WARC [32] are the established standard formats in Web archiving. Each file is a sequence of **records**. A record consists of a header and a content block. The header is a set of key-value pairs which are specific for every record type. There are eight record types: warcinfo, response, request, metadata, revisit, conversion, and continuation. The **response** record type is the one which stores archived pages and the accompanying metadata. Listing 7.1 shows an example of a WARC response record, taken from the WARC specification [32]. The headers provide information about the

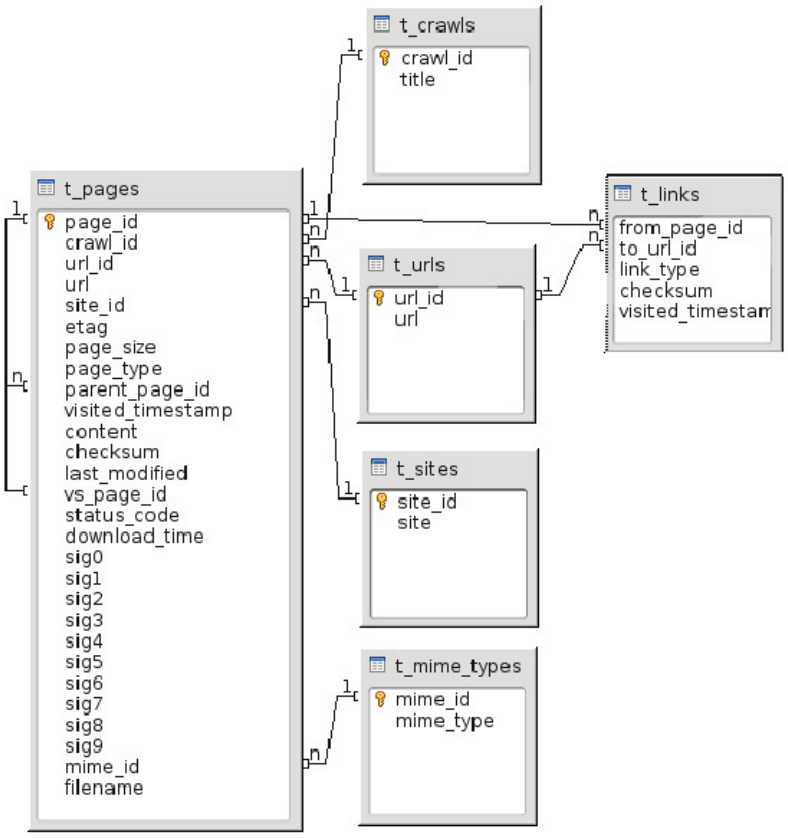


Figure 7.3.: Database Schema

URL, the time of creation, and the content of the record. The content block is an exact copy of a received HTTP response.

```
warc/0.9 7583 response http://www.archive.org/images/logo.jpg
    20050708010101 message/http
    uuid:a4b26b6b-f918-4136-af04-f859d75aebe5
IP-Address: 207.241.224.241
Related-Record-ID: uuid:f569983a-ef8c-4e62-b347-295b227c3e51
Checksum: sha1:2ZWC6JAT6KNXKD37F7MOEKXQMRY75YY4

HTTP/1.x 200 OK
Date: Fri, 08 Jul 2005 01:01:01 GMT
Server: Apache/1.3.33 (Debian GNU/Linux) PHP/5.0.4-0.3
Last-Modified: Sun, 12 Jun 2005 00:31:01 GMT
Etag: "914480-1b2e-42ab8245"
Accept-Ranges: bytes
Content-Length: 6958
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: image/jpeg

[6958 bytes of binary data here]
```

Listing 7.1: WARC example

Our prototype parses the records in a WARC or in an ARC file and maps the headers to the database schema. Unfortunately, neither ARC nor WARC formats support link information of the Web site. We obtain this information from the DAT files, conveniently created by the Heritrix crawler [65] during the archival and URL extraction process. If ARC and WARC files are available, then the link structure between the pages can be recreated with the help of the URL extraction module of Heritrix from the archived HTML pages.

Web archive data needs to be cleaned prior to any analysis. The most typical problem here are multiple downloads of the same page/URL. This occurs due to several reasons: some pages are downloaded many times to reflect the download policy of the Web site (such as robots.txt), since some embedded material was not available at the time of the download, or because pages might be re-downloaded by the archivist to improve the quality of the coverage/quality of the site. Furthermore, pages can be downloaded multiple times because of the pure formatting of the URLs of the pages: if the Web server does not distinguish upper and lower case in URLs, the designers of the individual pages tend to use different capitalizations of both the filenames and the subdirectories. Large parts of the site be be revisited multiple times for this reason. Removal of duplicates and cleansing is essential for such data, since different capitalization influences the number of changed documents and complicates the analysis of the history of changes for a given page.

SQL code for the removal of duplicates is given in the Appendix (cf. Listing A.1 for details). The algorithm identifies the tuple with the largest timestamp (cf. Lines 3–4) in all groups of pages with the same url (cf. Line 6), and filters out all other tuples (cf. Lines 7–10). Once the duplicates from the t_pages relation are removed, the duplicate links can be removed by removing all tuples that are not referenced in the cleaned t_pages relation. For convenience we store the cleaned data in t_pages_dd and t_links_dd relations (cf. Lines 1 and 12).

Removing duplicates of formatting of the URLs can be done similarly (cf. Listing A.2 in the Appendix). All URLs need to be lowercased, grouped by the same URLs, and the URL with the latest timestamp is taken. However such an approach involves many and costly string comparisons. Instead, we establish the grouping on the ids of the URLs level (cf. Lines 1–8), and compute the smallest values for each group (cf. Lines 10–33).

7.3. Conclusions

The prototype we developed was used in practice for Web archiving by our partners in the Living Web Archives project. This proved the applicability of the proposed Web archiving model and crawling strategies. Furthermore, the data stored in the database from both Web crawls and ARC/WARC files helped analyzing the changes of Web pages.

In the next chapter, we propose an approach for visual analysis of the coherence of Web archives.

Chapter 8.

Visual Analysis

8.1. Introduction

The visual analysis of Web archives serves to examine the coherence of the archives and to improve crawling strategies such that future crawl will be as coherent as possible.

According to our model Web archives may employ either single-visit strategies or visit-revisit strategies to capture Web sites. We can compute only the coherence of visit-revisit captures. Although coherence for single-visit captures is not formally defined, we can still reason about it. If we pair two sequential single-visit captures, we can simulate visits and revisits. In a series of single-visit captures, this helps to identify site captures which are potentially of lower quality than the average.

Visual analysis of Web archives not only makes possible estimating the coherence of single-visit captures but also gives insights for adjustment of the crawling strategies. If changed Web pages are clustered in a subsite of the Web site, then the archivist may decide to crawl this particular subsite separately. Conversely, if a subsite contains only pages which did not change and are unlikely to change, the archivist may exclude the subsite from the next crawls.

For the aforementioned purposes, we suggest three techniques for visual analysis of changes in site captures:

- time series analysis of changes with area plots,
- analysis of changes with scatterplots,
- analysis of changes with graph visualization.

The first technique identifies single-visit captures with potentially low quality. The

other two techniques discover correlations among the types and the locations of the changed page in visit-revisit captures or in pairs of single-visit captures.

We choose single-visit captures of the Web site `sabre.mod.uk` from the UKGOV dataset and a visit-revisit capture from the MPII dataset to demonstrate the visualization techniques on both single-visit captures and visit-revisit captures.

The chapter is organized in the following way. Section 8.2 introduces the mechanisms for change detection in single-visit and visit-revisit captures. Sections 8.3–8.5 present the three visualization techniques and analyze the coherence in the selected site captures. Finally, Section 8.6 summarizes the chapter.

8.2. Datasets and Change Detection

In this section we describe the datasets selected for the visual analysis and the mechanisms to identify page changes.

We select single-visit captures of the Web site `sabre.mod.uk` from the UKGOV dataset and a visit-revisit capture from the MPII dataset.

The archive of the `sabre.mod.uk` Web site consists of 130 weekly single-visit captures. Altogether there are 2956 pages, 1809 are HTML pages, 1111 are images, and the rest are PDF files. Note, that the captures do not necessarily contain all the pages, since some pages are added and other removed during the timespan of the captures. We use this Web archive to demonstrate how archivists can reason about the coherence of single-visit captures by using of area plots and then visually compare coherent captures with less coherent ones.

The visit-revisit capture from the MPII dataset contains 61254 pages each one with a visit and a revisit. If the page has been removed between the visit and the revisit we store the HTTP response from the Web server indicating that the page is not online anymore. We use this capture to demonstrate how archivits can identify subsites of the Web site for separate crawls or for exclusion from next crawls. To this end, we use graph visualization which we discuss in detail in Section 8.5.

We detect changes in captured Web pages using one of the following techniques: conditional GET requests, comparison of fingerprints of two versions of a Web page, or computing the similarity between two versions with *shingles*.

Checking for changes with conditional GET requests takes place during the crawl. When the crawler requests for a page, it may send a fingerprint of an already captured

version of the Web page. The Web server will deliver the page only if the page has changed after its previous download. If there are no changes, the crawler will store only the HTTP response from the server without actually downloading the Web page. However, the conditional GET mechanism is optional according to the HTTP protocol [40] and not all Web servers may support it. In this case, we check for changes in a Web page using a two captured versions of it.

Since byte-wise comparison between two versions of the same page may be time-consuming, we compare the fingerprints of the two versions. At every visit, the crawler gets not only the current content of the page, but also a 16-byte long fingerprint generated by the MD5 Message-Digest Algorithm [45]. Any change in the content of the page will be mirrored in the fingerprint. Sometimes, however, we consider two pages with negligible differences as identical. For example, it is reasonable to consider two versions of a Web page identical if the only difference is a timestamp indicating the retrieval time. In this case, we compute the similarity of versions using shingles [16].

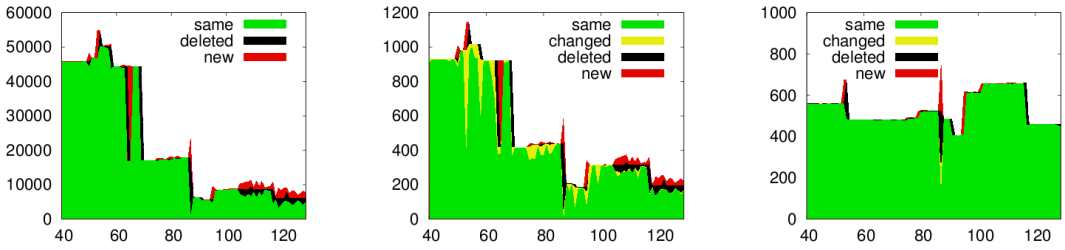
Shingles are n-gram fragments on word level. If a set of shingles represents each version of a Web page, then the similarity between two versions is the Jaccard distance between the corresponding sets of shingles. To compute of the shingles we consider only the text content of the Web pages. This ensures that flash contents change at every visit of the page (like advertisement banners or visit counters) are filtered out. Since we want to take into account all other changes, we work with 10-grams and a high similarity threshold (0.9). The *temporal shingling* [84] introduced by Schenkel can be also applied if visits of the Web pages are frequent enough (every two minutes).

Once we have detected the changes in the Web pages, we can proceed with the visual analysis of the site captures.

8.3. Time Series Analysis with Area Plots

We use time series analysis to detect captures with potentially low coherence. Area plots visualize the difference between every two subsequent captures of the same Web site. The magnitude of the difference indicates possible incoherence of the earlier capture.

he series of the area plots (cf. Figures 8.1a– 8.1c) can be used to get an overview of the percentage of change in the captures of the `sabre.mod.uk` Web site as the time passes. The X axis shows the id of the crawl (which increases with time), while the Y axis shows the number of changed/unchanged/new/deleted pages. We draw separate figures for links



(a) Links (b) HTML pages (c) Images

Figure 8.1.: Crawl sequence analysis of `www.sabre.mod.uk` site

(cf. Figure 8.1a), HTML pages (cf. Figure 8.1b), images (cf. Figure 8.1c).

Archivists can look for captures that contain significant number of changes. For example, one can see that there are significant changes in the Web site in captures 52 and 93. Also the Web site itself greatly shrank over time. In that respect, the area plots of links and HTML pages are helpful, while the area plot of images does not show the decreasing trend.

Computation of area graphs can be expressed in pure SQL and optimized by the query optimizer (cf. Listing B.1 in the Appendix) resulting in the overall $O(n \log n)$ or better complexity.

8.4. Change Analysis with Scatterplots

Three-dimensional scatterplots visualize changes in a capture revealing corelations among the changed pages. The visualization also helps to take a deeper look at the captures with lower quality than average. Archivists can see where problems may arise and to what extend the capture is applicable for their purposes.

Figures 8.2- 8.3 show scatterplot visualizations for the `sabre.mod.uk` site. We selected captures 53 and 54 as examples. We have detected that the Capture 53 has many changes while the next one contains fewer changes and is therefore more coherent. The scatterplots give reveal details about the difference in quality between the two captures.

The scatterplot visualizations map the mime-type, the first-order subdirectory of the and the URL of the page to the X, Y, and Z axes of the three-dimensional cube, while the color shows whether the change took place (new pages are colored red, changed pages are colored yellow, while the green color depicts unchanged pages). The archivist should look for patterns of changed and added pages. For example from Figure 8.2 one can see

that there are a few changes in the HTML files in the *output* and *textonly* subdirectories (the yellow points in the figure) and a whole new subdirectory of files are added into the Web archive (the red points at the top of the visualization). The changes of the pages (cf. the four yellow points) indicate the dependencies between the pages: if a page changes in the *output* directory then the corresponding page will change in the *textonly* directory. The newly added pages show that the Web site underwent significant changes in terms of structure, though content-wise the site did not vary much. Very few additional pages were introduced in Capture 54 indicating a high quality of the archive.

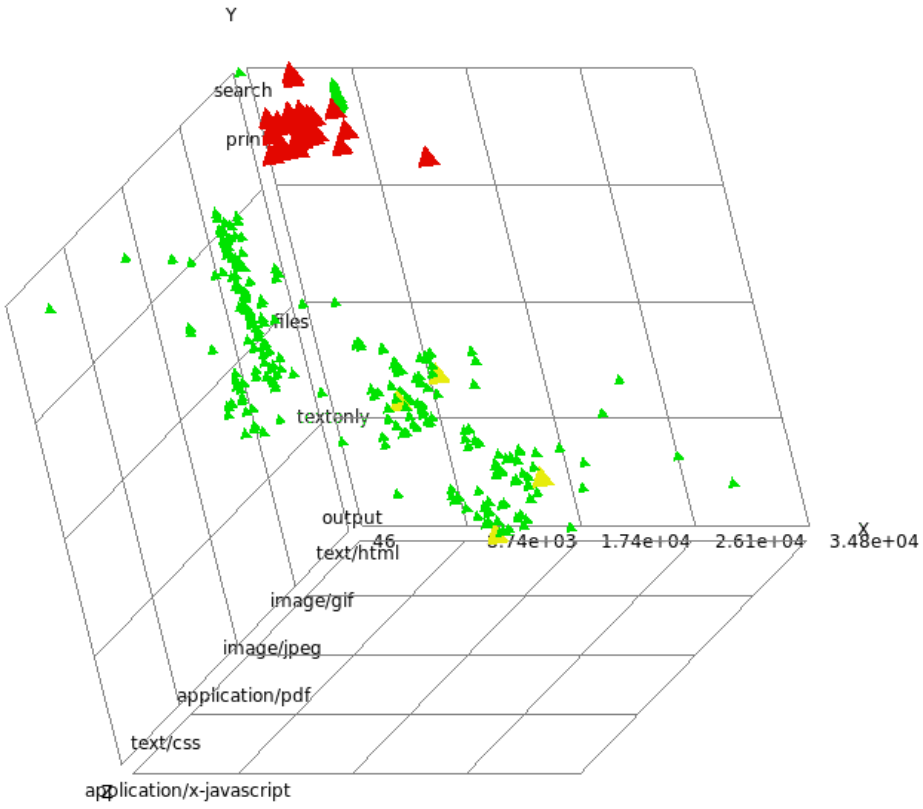


Figure 8.2.: Change analysis of Capture 53

Computation of the visualization is similar to the one of the Areal Plots (cf. Section 8.3).

8.5. Change Analysis with Graph Visualization

Visualization of site captures not only can reveal patterns of changes like in the previous section but also identify internal subsites in the Web sites.

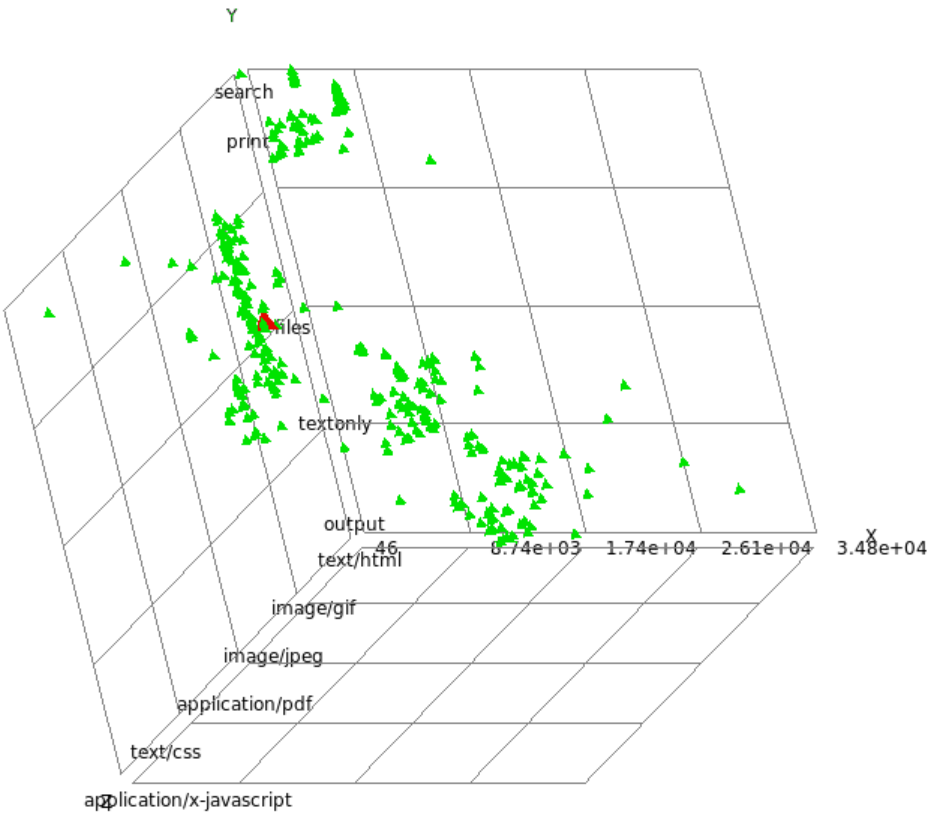


Figure 8.3.: Change analysis of Capture 54

Many big Web sites have internal subsites which follow specific change patterns. Knowing this in advance will help archivists plan better future crawls, for example they may specify that pages whose URLs are of certain pattern will be crawled separately or even be skipped from crawling. The site hierarchy inferred from URLs of the Web pages or sitemaps provided by the site owners are good indicators for internal structures. However, there are questions which we can not answer using only URLs or sitemaps. For example: Do the directories *output* and *textonly* of the Web site `sabre.mod.uk` (cf. Figure 8.2) form a separate subsite or not? Looking only at the URLs of the Web pages we will not consider the pages related. If we see that these pages change in a similar fashion we may decide otherwise. We want to bring such patterns to the attention of the archivists so that they make informed decision about the scope of the next crawls balancing between the coverage and the coherence of the captures.

In this section we present an approach how to visualize captures as graphs and how to analyze them for possible internal structures. From the huge amout of information

available from the capture we filter out and display only the information about the changes in the capture. By choosing graph representation we make sure that the changed pages are in close proximity so patterns can emerge. For that purpose, we choose a (spanning) tree representation derived from the crawler's path within the site.

Visualizing the spanning tree gives insights about the position and the nature of the changes in the Web contents compared with a previous crawl. However, the spanning tree usually is large in size and infeasible for many visualization tools. To overcome that problem and to focus on changes, we compress the tree and visualize only its relevant components.

Algorithm 7 collapseNode

```
Require: Node node
  node.collapsing=true
  if hasLinkChange(node) then
      node.color=red
      node.collapsing=false
  else if hasContentChange(node) then
      node.color=yellow
      node.collapsing=false
  else
      node.color=green
      for all children of node do
          collapseNode(child)
          if child.collapsing=false then
              node.collapsing=false
          else
              node.collapsingSize=child.collapsingSize+1
          end if
      end for
  end if
```

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns/graphml" xmlns:xsi="http://www.w3
    .org/2001/XMLSchema-instance" xmlns:y="http://www.yworks.com/xml/graphml"
    xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns/graphml http://www.yworks
    .com/xml/schema/graphml/1.0/ygraphml.xsd">
 ...
 <graph edgedefault="directed" id="G229">
  <node id="http://www.mpi-inf.mpg.de/index.html">
   <data key="d0">
    <y:ShapeNode>
     <y:Geometry width="10.003" height="10.003"/>
     <y:Fill color="#00FF00" transparent="false"/>
     <y:Shape type="ellipse"/>
    </y:ShapeNode>
   </data>
   <data key="d1">http://www.mpi-inf.mpg.de/index.html OK</data>
  </node>
  ...
  <edge source="http://www.mpi-inf.mpg.de/index.html" target="dns:www.mpi-inf.mpg.de"/>
  ...
 </graph>
</graphml>
```

Listing 8.1: Coherence defect graphML-file (excerpt)

In the first step, we analyze the nodes of the spanning tree and "flag" them according to their status: green if they are unchanged (coherent), yellow in case of text changes only, red in case of link changes, and, finally, black if a page is missing in the revisit phase or in the subsequent crawl.

Afterwards, we apply a collapsing strategy (cf. Algorithm 7), which tags the nodes in each fully coherent subtree as collapsible and marks the changed nodes and their ancestors as non-collapsible. In the visualization step (cf. Algorithm 8) we draw the non-collapsible nodes colored as detected. Additionally, for each collapsed subtree we draw a node proportional in size to the number of the nodes in the subtree and connect this node to the parent of the subtree.

For a graphical representation, the previously computed compressed tree representation is exported into a graphML-file (cf. Listing 8.1). The graphML [43] format is an XML-based standard and a file format for graphs. It describes the structural properties of the nodes and applied in many graph related software applications (yFiles[1], visone[2]).

We chose a visit-revisit capture from our MPII dataset to demonstrate the advantages of the graph visualization. We export the data into graphML format and visualize it with

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

[2] http://www.visone.info

Algorithm 8 drawNode

```
Require: Node node
  paintNode(node)
  for all children of node do
    if child.collapsing=false then
      drawNode(child)
      paintEdge(node, child)
    end if
  end for
  if node.collapsingSize > 0 then
    create node collapsedChild with size node.collapsingSize and green color
    paintNode(collapsedChild)
    paintEdge(node, collapsedChild)
  end if
```

a compliant software.

Figure 8.4 depicts a sample visualization of an `mpi-inf.mpg.de` capture (61254 pages). We observe that the majority of changed pages are clustered in two subtrees. The two clusters with changed pages consist of Web pages from two internal content management systems employed by different departments of our institute. The content management systems provide templates for both content and link structure. This explains the patterns visible on the figure. The first cluster, most of the changes form triples with two changed pages and a unchanged one. The two changed pages are on higher level in the subtree than the unchanged page. In the second cluster, the pattern is different. The pages again form triples, but this time with two unchanged pages and a changed page between them. An archivist provided with such information about a capture may decide to crawl these pages separately from the rest or may relax her requirements on changes, so that the changes from content management systems do not affect the quality of the capture.

At the same time, nodes much bigger in size than the rest represent many unchanged pages. The largest node represents almost one third of all pages 19712, the second largest node represents 4304 pages. Both of the nodes contain teaching material for courses from last semesters. The material is archived and the Web pages are not expected to change. Given such knowledge, the archivist may decide that future crawls do not visit these pages. By reducing by a third the number of pages in the capture, the chances to capture

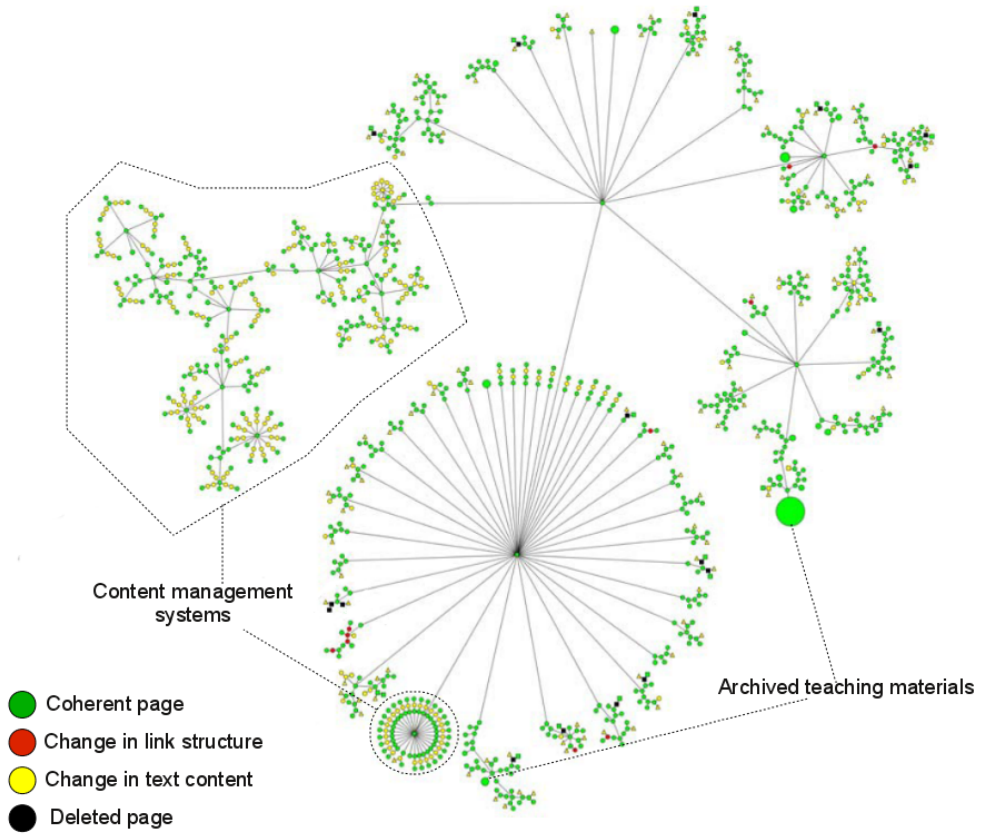


Figure 8.4.: Graph visualization of a visit-revisit capture of `mpi-inf.mpg.de`

the other pages increases.

8.6. Conclusions

In this chapter, we presented visualization techniques which complement our approach on Web archive quality. We extend the coherence analysis to single-visit captures, identify change patterns, and discover subsites supported by content management systems.

In Chapter 3 we defined coherence as a quality measure for visit-revisit captures. Here, we demonstrated how to reason about coherence in single-visit captures using area plots of series of captures. Furthermore, we investigated the change patterns with the help of scatterplots. Finally, the visualization of a capture as a tree (the spanning tree of the crawl) clustered together Web pages from content management systems.

Chapter 9.

Conclusions

Data quality is crucial for the future exploitation of Web archives. Similarly to traditional archives, Web archive applications cover both exploratory and evidentiary usage. We developed a comprehensive model for Web archiving that encompasses both cases. An appropriate quality measure was defined for each case: blur for exploratory usage, coherence for evidentiary usage. We proposed a framework of quality-conscious crawling strategies which optimize for either of the measures. The family of single-visit strategies minimizes the blur and the family of visit-revisit strategies maximizes the coherence. The experimental evaluation of the strategies confirmed that they outperform their competitors. Since the download order of the pages depends on the change behaviour of the Web pages, we adopted the Poisson process with page-specific change rates for modeling the changes. Furthermore, we showed that these rates can be statistically predicted based on various page properties. A prototype built on top of the Heritrix Web crawler demonstrated the applicability of the developed strategies. Finally, we proposed three visualization techniques to highlight the changes in a site capture. Thus, an archivist is able to refine the scope of future crawls in order to get high-quality captures.

Future extensions of the model and the strategies presented in this thesis can go into various directions. First, we can include automatic detection of subsites. For that purpose, we suggested several visualization techniques to the user. However, an automatic subsite detection will speed up scope refinement resulting in captures of higher quality. We could detect subsites from sitemaps, Web site hierarchies, or change patterns.

Another possible direction is to assign priorities to Web pages. This will be a minor change to our mathematical model. If we interpret the priority as the probability that an archived Web page will be in the result set of a time-travel query, then the priority will be a factor modifying the blur or the coherence of the page. A challenging task,

however, is to assign the priorities. Certainly, sitemaps provide specific values but they do not necessarily reflect the preferences of the Web site visitors. Furthermore, users' interests change with time. For example, the general public will not be interested in a specialized Web site commenting the strategic decisions of a businessperson. However, if this businessperson runs for the office of a country's president, all her previous decisions and statements will be put under heavy scrutiny. As a result the priority of the specialized Web site will increase dramatically. How to include such shifts in the society interests in the Web archiving model is an open question.

Continuous crawling is yet another option for future development. In this case, our Web archiving model have to be completely revised. A Web archive is not a series of periodic captures anymore. Results of time-travel queries will be expressed in different terms, not as site captures. Both blur and coherence will have different mathematical definitions.

The last direction we point out is benchmarking. A benchmarking dataset would steer further research towards the best practices in Web archiving. However, designing a benchmarking dataset would entail coordination with the Web archiving institutions and the Web content publishers.

Bibliography

[1] Serge Abiteboul, Mihai Preda, and Gregory Cobena. Adaptive on-line page importance computation. In *Proceedings of the 12th international conference on World Wide Web*, WWW '03, pages 280–290, New York, NY, USA, 2003. ACM.

[2] Eytan Adar, Mira Dontcheva, James Fogarty, and Daniel S. Weld. Zoetrope: interacting with the ephemeral Web. In *Proceedings of the 21st annual ACM symposium on User interface software and technology*, UIST '08, pages 239–248, New York, NY, USA, 2008. ACM.

[3] Eytan Adar, Jaime Teevan, Susan T. Dumais, and Jonathan L. Elsas. The Web changes everything: understanding the dynamics of Web content. In *Proceedings of the 2nd ACM international conference on Web search and data mining*, WSDM '09, pages 282–291, New York, NY, USA, 2009. ACM.

[4] Charu C. Aggarwal, Fatima Al-Garawi, and Philip S. Yu. Intelligent crawling on the World Wide Web with arbitrary predicates. In *Proceedings of the 10th international conference on World Wide Web*, WWW '01, pages 96–105, New York, NY, USA, 2001. ACM.

[5] Avishek Anand, Srikanta Bedathur, Klaus Berberich, Ralf Schenkel, and Christos Tryfonopoulos. EverLast: A distributed architecture for preserving the Web. In *Proceedings of the 9th ACM/IEEE joint conference on Digital libraries*, JCDL'09, pages 331–340, Austin, Texas, 2009. ACM.

[6] North Carolina State Archives and State Library of North Carolina. North Carolina State Government Web site archives. `http://webarchives.ncdcr.gov/`, April 2012.

[7] The National Archives. UK Government Web archive. `http://www.nationalarchives.gov.uk/webarchive/`, April 2012.

[8] Ricardo Baeza-Yates and Carlos Castillo. Balancing volume, quality and freshness in Web crawling. In Ajith Abraham, Javier Ruiz-del-Solar, and Mario Köppen, editors, *Soft Computing Systems - Design, Management and Applications*, Frontiers in Artificial Intelligence and Applications, pages 565–572. IOS Press, 2002.

[9] Ricardo Baeza-Yates, Carlos Castillo, Mauricio Marin, and Andrea Rodriguez. Crawling a country: better strategies than breadth-first for Web page ordering. In *Special interest tracks and posters of the 14th international conference on World Wide Web*, WWW '05, pages 864–872, New York, NY, USA, 2005. ACM.

[10] Luciano Barbosa and Juliana Freire. An adaptive crawler for locating hidden-Web entry points. In *Proceedings of the 16th international conference on World Wide Web*, WWW '07, pages 441–450, New York, NY, USA, 2007. ACM.

[11] Myriam Ben Saad and Stéphane Gançarski. Archiving the Web using page changes patterns: a case study. In *Proceedings of the 11th annual international ACM/IEEE joint conference on Digital libraries*, JCDL '11, pages 113–122, New York, NY, USA, 2011. ACM.

[12] András A. Benczúr, Dávid Siklósi, Jácint Szabó, István Biró, Zsolt Fekete, Miklós Kurucz, Attila Pereszlényi, Simon Racz, and Adrienn Szabo. In *Proceedings of the 8th international Web archiving workshop*, IWAW '08.

[13] Michael Bendersky, W. Bruce Croft, and Yanlei Diao. Quality-biased ranking of Web documents. In *Proceedings of the 4th ACM international conference on Web search and data mining*, WSDM '11, pages 95–104, New York, NY, USA, 2011. ACM.

[14] Brian E. Brewington and George Cybenko. Keeping up with the changing Web. *Computer*, 33(5):52–58, May 2000.

[15] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual Web search engine. In *Proceedings of the 7th international conference on World Wide Web*, WWW '98, pages 107–117, Amsterdam, The Netherlands, The Netherlands, 1998. Elsevier Science Publishers B. V.

[16] Andrei Z. Broder, Steven C. Glassman, Mark S. Manasse, and Geoffrey Zweig. Syntactic clustering of the Web. In *Selected papers from the 6th international*

conference on World Wide Web, WWW '97, pages 1157–1166, Essex, UK, 1997. Elsevier Science Publishers Ltd.

[17] Niels Brügger. *Archiving Web sites: general considerations and strategies*. The Centre for Internet Research, 2005.

[18] Niels Brügger. The archived Web site and Web site philology: a new type of historical document? *Nordicom Review*, 29(2):151–171, 2008.

[19] Deutscher Bundestag. Deutscher Bundestag: Web-Archiv. `http://webarchiv.bundestag.de/cgi/kurz.php`, April 2012.

[20] Soumen Chakrabarti, Martin van den Berg, and Byron Dom. Focused crawling: A new approach to topic-specific Web resource discovery. *Computer Networks*, 31(11-16):1623–1640, 1999.

[21] Ling Chen, Sourav S. Bhowmick, and Wolfgang Nejdl. NEAR-Miner: mining evolution associations of Web site directories for efficient maintenance of Web archives. *Proceedings of the VLDB Endowment*, 2(1):1150–1161, August 2009.

[22] Ed H. Chi, James Pitkow, Jock Mackinlay, Peter Pirolli, Rich Gossweiler, and Stuart K. Card. Visualizing the evolution of Web ecologies. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '98, pages 400–407, New York, NY, USA, 1998. ACM Press/Addison-Wesley Publishing Co.

[23] Junghoo Cho and Hector Garcia-Molina. The evolution of the Web and implications for an incremental crawler. In *Proceedings of the 26th international conference on Very large data bases*, VLDB '00, pages 200–209, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[24] Junghoo Cho and Hector Garcia-Molina. Synchronizing a database to improve freshness. In *Proceedings of the 19th ACM SIGMOD international conference on Management of data*, SIGMOD '00, pages 117–128, New York, NY, USA, 2000. ACM.

[25] Junghoo Cho and Hector Garcia-Molina. Effective page refresh policies for Web crawlers. *ACM Transactions on Database Systems*, 28(4):390–426, December 2003.

[26] Junghoo Cho and Hector Garcia-Molina. Estimating frequency of change. *ACM Transactions on Internet Technology*, 3(3):256–290, August 2003.

[27] Junghoo Cho, Hector Garcia-Molina, and Lawrence Page. Efficient crawling through URL ordering. In *Proceedings of the 7th international conference on World Wide Web*, WWW' 98, pages 161–172, Amsterdam, The Netherlands, The Netherlands, 1998. Elsevier Science Publishers B. V.

[28] Junghoo Cho, Sourashis Roy, and Robert E. Adams. Page quality: in search of an unbiased Web ranking. In *Proceedings of the 24th ACM SIGMOD international conference on Management of data*, SIGMOD '05, pages 551–562, New York, NY, USA, 2005. ACM.

[29] Junghoo Cho and Uri Schonfeld. RankMass crawler: a crawler with high personalized pagerank coverage guarantee. In *Proceedings of the 33rd international conference on Very large data bases*, VLDB '07, pages 375–386. VLDB Endowment, 2007.

[30] Edward G. Coffman, Zhen Liu, and Richard R. Weber. Optimal robot scheduling for Web search engines. *Journal of Scheduling*, 1(1):15–29, 1998.

[31] International Internet Preservation Consortium. ARC_IA, internet archive ARC file format. `http://www.digitalpreservation.gov/formats/fdd/fdd000235.shtml`.

[32] International Internet Preservation Consortium. WARC, Web archive file format. `http://www.digitalpreservation.gov/formats/fdd/fdd000236.shtml`.

[33] Dimitar Denev, Arturas Mazeika, Marc Spaniol, and Gerhard Weikum. Sharc: framework for quality-conscious Web archiving. *Proceedings of the VLDB Endowment*, 2(1):586–597, August 2009.

[34] Dimitar Denev, Arturas Mazeika, Marc Spaniol, and Gerhard Weikum. The SHARC framework for data quality in Web archiving. *The VLDB Journal*, 20(2):183–207, April 2011.

[35] Stephen Dill, Ravi Kumar, Kevin S. Mccurley, Sridhar Rajagopalan, D. Sivakumar, and Andrew Tomkins. Self-similarity in the Web. *ACM Transactions on Internet Technology*, 2(3):205–223, August 2002.

[36] Anlei Dong, Yi Chang, Zhaohui Zheng, Gilad Mishne, Jing Bai, Ruiqiang Zhang, Karolina Buchner, Ciya Liao, and Fernando Diaz. Towards recency ranking in Web search. In *Proceedings of the third ACM international conference on Web search and data mining*, WSDM '10, pages 11–20, New York, NY, USA, 2010. ACM.

[37] Fred Douglis, Anja Feldmann, Balachander Krishnamurthy, and Jeffrey Mogul. 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*, USITS'97, pages 14–14, Berkeley, CA, USA, 1997. USENIX Association.

[38] Miklós Erdélyi, András A. Benczúr, Julien Masanés, and Dávid Siklósi. Web spam filtering in internet archives. In *Proceedings of the 5th international workshop on Adversarial information retrieval on the Web*, AIRWeb '09, pages 17–20, New York, NY, USA, 2009. ACM.

[39] Dennis Fetterly, Mark Manasse, Marc Najork, and Janet Wiener. A large-scale study of the evolution of Web pages. In *Proceedings of the 12th international conference on World Wide Web*, WWW '03, pages 669–678, New York, NY, USA, 2003. ACM.

[40] Roy Fielding, James Gettys, Jeffrey C. Mogul, Henrik Nielsen, Larry Masinter, Paul J. Leach, and Tim Berners-Lee. RFC 2616: Hypertext Transfer Protocol – HTTP/1.1. `http://tools.ietf.org/html/rfc2616`, 1999.

[41] Foundation for National Scientific Computing. Portuguese Web archive: search the past. `http://www.archive.pt/`, April 2012.

[42] Daniel Gomes, João Miranda, and Manuel Costa. A survey on web archiving initiatives. In Stefan Gradmann, Francesca Borri, Carlo Meghini, and Heiko Schuldt, editors, *Research and Advanced Technology for Digital Libraries*, volume 6966 of *Lecture Notes in Computer Science*, pages 408–420. Springer Berlin / Heidelberg, 2011. 10.1007/978-3-642-24469-8_41.

[43] GraphML Working Group. The graphML file format. `http://graphml.graphdrawing.org/`, 2002.

[44] IIPC Access Working Group. Use cases for access to Internet archives. `http://www.netpreserve.org/publications/iipc-r-003.pdf`, 2006.

[45] Network Working Group. The MD5 Message-Digest Algorithm. `http://tools.ietf.org/html/rfc1321`, 1992.

[46] Jiawei Han. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[47] Allan Heydon and Marc Najork. Mercator: A scalable, extensible Web crawler. *World Wide Web*, 2(4):219–229, April 1999.

[48] Masahiko Itoh, Masashi Toyoda, and Masaru Kitsuregawa. An interactive visualization framework for time-series of Web graphs in a 3D environment. In *Proceedings of the 14th international conference on Information visualisation*, IV '10, pages 54–60, Washington, DC, USA, 2010. IEEE Computer Society.

[49] Adam Jatowt, Yukiko Kawai, Satoshi Nakamura, Yutaka Kidawara, and Katsumi Tanaka. A browser for browsing the past Web. In *Proceedings of the 15th international conference on World Wide Web*, WWW '06, pages 877–878, New York, NY, USA, 2006. ACM.

[50] George Karypis and Eui-Hong (Sam) Han. Fast supervised dimensionality reduction algorithm with applications to document categorization & retrieval. In *Proceedings of the 9th international conference on Information and knowledge management*, CIKM '00, pages 12–19, New York, NY, USA, 2000. ACM.

[51] Sung Jin Kim and Sang Ho Lee. Estimating the change of Web pages. In *Proceedings of the 7th international conference on Computational Science, Part III*, ICCS '07, pages 798–805, Berlin, Heidelberg, 2007. Springer-Verlag.

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

[53] Martijn Koster. A standard for robot exclusion. `http://www.robotstxt.org/orig.html`, 1994.

[54] Columbia University Libraries. Human rights Web archive. `http://library.columbia.edu/indiv/humanrights/hrwa.html`, April 2012.

[55] Harvard University Library. Web archive collection service - Harvard University Library. `http://wax.lib.harvard.edu/collections/home.do`, April 2012.

[56] Bing Liu and Filippo Menczer. Web crawling. In *Web Data Mining*, Data-Centric Systems and Applications, pages 311–362. Springer Berlin Heidelberg, 2011.

[57] Ling Liu, Calton Pu, and Wei Tang. WebCQ-detecting and delivering information changes on the Web. In *Proceedings of the 9th international conference on Information and knowledge management*, CIKM '00, pages 512–519, New York, NY, USA, 2000. ACM.

[58] Yuting Liu, Bin Gao, Tie-Yan Liu, Ying Zhang, Zhiming Ma, Shuyuan He, and Hang Li. Browserank: letting Web users vote for page importance. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '08, pages 451–458, New York, NY, USA, 2008. ACM.

[59] Julien Masanès. *Web archiving*. Springer, 2006.

[60] Norman Matloff. Estimation of Internet file-access/modification rates from indirect data. *ACM Transactions on Modeling and Computer Simulation*, 15(3):233–253, July 2005.

[61] Arturas Mazeika, Dimitar Denev, Marc Spaniol, and Gerhard Weikum. The SOLAR system for sharp Web archiving. In Julien Masanès, Andreas Rauber, and Marc Spaniol, editors, *Proceedings of the 10th international Web archiving workshop*, IWAW '10, pages 24–30, 2010.

[62] Filippo Menczer. Lexical and semantic clustering by Web links. *Journal of the American Society for Information Science and Technology - Special issue: Webometrics*, 55(14):1261–1269, 2004.

[63] Filippo Menczer and Richard K. Belew. Adaptive retrieval agents: Internalizing local context and scaling up to the Web. *Machine Learning*, 39(2/3):203–242, 2000.

[64] Filippo Menczer, Gautam Pant, and Padmini Srinivasan. Topical Web crawlers: Evaluating adaptive algorithms. *ACM Transactions on Internet Technology*, 4(4):378–419, 2004.

[65] Gordon Mohr, Michael Stack, Igor Ranitovic, Dan Avery, and Michele Kimpton. Introduction to Heritrix, an archival quality Web crawler. In *Proceedings of the 4th international Web archiving workshop*, IWAW '04, 2004.

[66] Marc Najork and Janet L. Wiener. Breadth-first crawling yields high-quality pages. In *Proceedings of the 10th international conference on World Wide Web*, WWW '01, pages 114–118, New York, NY, USA, 2001. ACM.

[67] Alexandros Ntoulas, Petros Zerfos, and Junghoo Cho. Downloading textual hidden Web content through keyword queries. In *Proceedings of the 5th ACM/IEEE-CS joint conference on Digital libraries*, JCDL '05, pages 100–109, New York, NY, USA, 2005. ACM.

[68] National Library of Australia. Pandora Archive - Preserving and accessing networked documentary resources of Australia. `http://pandora.nla.gov.au/`, April 2012.

[69] Library of Catalonia. PADICAT, Patrimoni Digital de Catalunya. `http://www.padicat.cat/`, April 2012.

[70] University of Hawaii at Manoa Library. Web archiving project for the Pacific Islands. `http://library.manoa.hawaii.edu/research/archiveit/`, April 2012.

[71] National Library of Korea. OASIS (Online Archiving and Searching Internet Sources). `http://www.oasis.go.kr/intro_new/intro_overview_e.jsp`, April 2012.

[72] State Library of Tasmania. Our digital island. `http://odi.statelibrary.tas.gov.au/`, April 2012.

[73] University of Texas at Austin. Latin American Web archiving project, LAWAP. `http://lanic.utexas.edu/project/archives/`, April 2012.

[74] Christopher Olston and Marc Najork. Web crawling. *Foundations and Trends in Information Retrieval*, 4(3):175–246, 2010.

[75] Christopher Olston and Sandeep Pandey. Recrawl scheduling based on information longevity. In *Proceedings of the 17th international conference on World Wide Web*, WWW '08, pages 437–446, New York, NY, USA, 2008. ACM.

[76] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the Web. Technical report, Stanford University, 1999.

[77] Sandeep Pandey, Kedar Dhamdhere, and Christopher Olston. WIC: a general-purpose algorithm for monitoring Web information sources. In *Proceedings of the 30th international conference on Very large data bases*, VLDB '04, pages 360–371. VLDB Endowment, 2004.

[78] Sandeep Pandey, Krithi Ramamritham, and Soumen Chakrabarti. Monitoring the dynamic Web to respond to continuous queries. In *Proceedings of the 12th international conference on World Wide Web*, WWW '03, pages 659–668, New York, NY, USA, 2003. ACM.

[79] Zeynep Pehlivan, Myriam Ben-Saad, and Stéphane Gançarski. Vi-DIFF: understanding Web pages changes. In *Proceedings of the 21st international conference on Database and expert systems applications: Part I*, DEXA '10, pages 1–15, Berlin, Heidelberg, 2010. Springer-Verlag.

[80] practice.com. Debunking the Wayback machine. `http://practice.com/2008/12/29/debunking-the-wayback-machine`, 2008.

[81] Myriam Ben Saad and Stéphane Gançarski. Using visual pages analysis for optimizing Web archiving. In *Proceedings of the 2010 EDBT/ICDT Workshops*, EDBT '10, pages 43:1–43:7, New York, NY, USA, 2010. ACM.

[82] Myriam Ben Saad and Stéphane Gançarski. Improving the quality of Web archives through the importance of changes. In *Proceedings of the 22nd international conference on Database and expert systems applications - Volume Part I*, DEXA'11, pages 394–409, Berlin, Heidelberg, 2011. Springer-Verlag.

[83] Myriam Ben Saad, Zeynep Pehlivan, and Stéphane Gançarski. Coherence-oriented crawling and navigation using patterns for Web archives. In *Proceedings of the 15th international conference on Theory and practice of digital libraries: research and advanced technology for digital libraries*, TPDL'11, pages 421–433, Berlin, Heidelberg, 2011. Springer-Verlag.

[84] Ralf Schenkel. Temporal shingling for version identification in Web archives. In *Proceedings of the 32nd European conference on Advances in information retrieval*, ECIR'2010, pages 508–519, Berlin, Heidelberg, 2010. Springer-Verlag.

[85] Uri Schonfeld and Narayanan Shivakumar. Sitemaps: above and beyond the crawl of duty. In *Proceedings of the 18th international conference on World wide web*, WWW '09, pages 991–1000, New York, NY, USA, 2009. ACM.

[86] Arie Segev and Arie Shoshani. Logical modeling of temporal data. In *Proceedings of the 7th ACM SIGMOD international conference on Management of data*, SIGMOD '87, pages 454–466, New York, NY, USA, 1987. ACM.

[87] Ka Cheung Sia, Junghoo Cho, and Hyun-Kyu Cho. Efficient monitoring algorithm for fast news alerts. *IEEE Transactions on Knowledge and Data Engineering*, 19(7):950–961, July 2007.

[88] Kristinn Sigurdsson. Incremental crawling with Heritrix. In *Proceedings of the 5th international Web archiving workshop*, IWAW '05, 2005.

[89] Sanasam Ranbir Singh. Estimating the rate of Web page updates. In *Proceedings of the 20th international joint conference on Artifical intelligence*, IJCAI'07, pages 2874–2879, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[90] Sergej Sizov, Martin Theobald, Stefan Siersdorfer, and Gerhard Weikum. BINGO!: Bookmark-induced gathering of information. In *Proceedings of the 3rd International Conference on Web information systems engineering*, WISE '02, pages 323–332, Washington, DC, USA, 2002. IEEE Computer Society.

[91] Sergej Sizov, Martin Theobald, Stefan Siersdorfer, Gerhard Weikum, Jens Graupmann, Michael Biwer, and Patrick Zimmer. The BINGO! system for information portal generation and expert web search. In *Proceedings of the 1st biennial conference on Innovative data systems research*, CIDR '03, pages 69–80, 2003.

[92] Marc Spaniol, Dimitar Denev, Arturas Mazeika, Gerhard Weikum, and Pierre Senellart. Data quality in Web archiving. In *Proceedings of the 3rd workshop on Information credibility on the Web*, WICOW '09, pages 19–26, New York, NY, USA, 2009. ACM.

[93] Marc Spaniol, Arturas Mazeika, Dimitar Denev, and Gerhard Weikum. "catch me if you can": Visual analysis of coherence defects in web archiving. In Julien Masanes, Andreas Rauber, and Marc Spaniol, editors, *Proceedings of the 9th international Web archiving workshop*, IWAW '09, pages 27–37, 2009.

[94] Qingzhao Tan and Prasenjit Mitra. Clustering-based incremental Web crawling. *ACM Transactions on Information Systems*, 28(4):17:1–17:27, November 2010.

[95] Qingzhao Tan, Prasenjit Mitra, and C. Lee Giles. Designing clustering-based Web crawling policies for search engine crawlers. In *Proceedings of the 16th ACM conference on Information and knowledge management*, CIKM '07, pages 535–544, New York, NY, USA, 2007. ACM.

[96] Masashi Toyoda and Masaru Kitsuregawa. A system for visualizing and analyzing the evolution of the Web with a time series of graphs. In *Proceedings of the 16th ACM conference on Hypertext and hypermedia*, HYPERTEXT '05, pages 151–160, New York, NY, USA, 2005. ACM.

[97] Yafang Wang, Mingjie Zhu, Lizhen Qu, Marc Spaniol, and Gerhard Weikum. Timely yago: harvesting, querying, and visualizing temporal knowledge from Wikipedia. In *Proceedings of the 13th international conference on Extending database technology*, EDBT '10, pages 697–700, New York, NY, USA, 2010. ACM.

[98] Ou Wu, Yunfei Chen, Bing Li, and Weiming Hu. Evaluating the visual quality of Web pages using a computational aesthetic approach. In *Proceedings of the 4th ACM international conference on Web search and data mining*, WSDM '11, pages 337–346, New York, NY, USA, 2011. ACM.

[99] Ping Wu, Ji-Rong Wen, Huan Liu, and Wei-Ying Ma. Query selection techniques for efficient crawling of structured Web sources. In *Proceedings of the 22nd international conference on Data engineering*, ICDE '06, pages 47–, Washington, DC, USA, 2006. IEEE Computer Society.

[100] Shuyi Zheng, Pavel Dmitriev, and C. Lee Giles. Graph-based seed selection for Web-scale crawlers. In *Proceedings of the 18th ACM conference on Information and knowledge management*, CIKM '09, pages 1967–1970, New York, NY, USA, 2009. ACM.

[101] Zhaohui Zheng, Hongyuan Zha, Tong Zhang, Olivier Chapelle, Keke Chen, and Gordon Sun. A general boosting method and its application to learning ranking functions for Web search. In *Proceedings of the 21st annual conference on Neural information processing systems*, NIPS '07, 2007.

Appendix A.

Prototype Implementation

```
create table t_pages_dd as
select t_pages.* from (
  select crawl_id, url_id, max(visited_timestamp)
         as latest_timestamp
  from t_pages
  group by crawl_id, url_id) as x, t_pages
where t_pages.crawl_id = x.crawl_id and
      t_pages.url_id = x.url_id and
      t_pages.visited_timestamp
           = x.latest_timestamp

create table t_links_dd as
select t_links.*
from t_pages_dd, t_links
where t_pages_dd.url_id = t_links.from_url_id and
      t_pages_dd.visited_timestamp
        = t_links.visited_timestamp
```

Listing A.1: Deduplication

```
create table clean_mapping as
select dirty_url.url_id as dirty_url_id,
       clean_url.url_id as clean_url_id
from (
  select min(url_id) as url_id, lower(url) as url
  from t_urls group by lower(url)
) as clean_url, t_urls as dirty_url
where clean_url.url = lower(dirty_url.url);

create table lower_url_dd as
select  crawl_id,
  clean_mapping.clean_url_id as url_id,
  lower(min(url)) as url, site_id as site_id,
  min(etag) as etag,
  min(page_size) as page_size,
  min(page_type) as page_type,
  min(visited_timestamp) as visited_timestamp,
  min(checksum) as checksum,
  min(last_modified) as last_modified,
  min(status_code) as status_code,
  min(download_time) as download_time,
  min(sig0) as sig0, min(sig1) as sig1,
  min(sig2) as sig2, min(sig3) as sig3,
  min(sig4) as sig4, min(sig5) as sig5,
  min(sig6) as sig6, min(sig7) as sig7,
  min(sig8) as sig8, min(sig9) as sig9,
  min(mime_id) as mime_id,
  min(filename) as filename
from t_pages_dd, clean_mapping
where t_pages_dd.url_id
      = clean_mapping.dirty_url_id
group by t_pages_dd.crawl_id, t_pages_dd.site_id,
      clean_mapping.clean_url_id
```

Listing A.2: SQL for cleaning to lower urls

Appendix B.

Visual Analysis

```
select XXX as crawl_id, site_id,
  coalesce(deleted,0)+coalesce(nnew,0) +
  coalesce(same,0)+coalesce(changed,0) as nall,
  deleted, nnew, same, changed from (
  select site_id,
   max(deleted) as deleted, min(nnew) as nnew,
   min(same) as same, min(changed) as changed
   from (
   select site_id,
   case when status='deleted' then count
        end as deleted,
   case when status='new' then count end as nnew,
   case when status='same' then count end as same,
   case when status='changed' then count
        end as changed from (
     select ssite_id as site_id, status,
     count(site_id) from (
       select
       case
         when nprev.site_id is null
           then ncurr.site_id
           else nprev.site_id
         end as ssite_id,
       case
         when nprev.site_id is null then 'new'
         when ncurr.site_id is null then 'deleted'
         when nprev.new_check_sum
           = ncurr.new_check_sum then 'same'
         else 'changed'
         end as status
       from (
         select site_id,url_id,new_check_sum
         from nodes
         where crawl_id = XXX-1
       ) as nprev
       full outer join (
         select site_id,url_id,new_check_sum
```

```
38          from nodes
39          where crawl_id = XXX
40        ) as ncurr
41        on nprev.site_id = ncurr.site_id and
42        nprev.url_id = ncurr.url_id
43      ) as crawls_with_status
44      group by ssite_id, status
45    ) as crosstab_multiple_rows
46  ) as crosstab_single_row
47  group by site_id
48 ) as aggregated
```

Listing B.1: SQL for area bars

List of Figures

List of Tables

List of Algorithms