"Catch me if you can": Visual Analysis of Coherence Defects in Web Archiving

Marc Spaniol, Arturas Mazeika, Dimitar Denev, and Gerhard Weikum
Max-Planck-Institut für Informatik
Campus E1 4
Saarbrücken, Germany
{mspaniol|amazeika|ddenev|weikum}@mpi-inf.mpg.de

ABSTRACT

The World Wide Web is a continuously evolving network of contents (e.g. Web pages, images, sound files, etc.) and an interconnecting link structure. Hence, an archivist may never be sure if the contents collected so far are still consistent with those contents she needs to retrieve next. Therefore, questions arise about detecting, measuring them and – finally – understanding coherence defects. To this end, visualization strategies are being presented that might be applied on different level of granularities: working with (in the ideal case) properly set last-modified timestamps, based on metadata extracted from the crawler in accelerated crawl-revisit pairs, or from the Internet Archive's WARC files. In order to help the archivist in understanding the nature of these defects, this paper investigates means for visualizing change behavior and archive coherence.

Categories and Subject Descriptors

H.3 [**Information Storage and Retrieval**]: Content Analysis and Indexing

General Terms

Measurement

Keywords

Web Archiving, Data Quality, Coherence Analysis and Visualization

1. INTRODUCTION

"Catch me if you can" is the title of a movie based on a true story about impostor Frank Abagnale Jr., who plays being a pilot, doctor, and lawyer. While the movie describes the difficulties in catching a real-life pretender, this situation can be somewhat compared with the problems in Web archiving: The World Wide Web enables millions of users to publish, change, and delete content on the Web. Like chasing an impostor, preservation and archival of the digital born media is not trivial and can contain data quality issues. For example, consider a content-management system (CMS) maintaining a Web site of a research organization: whenever, two researchers co-author a publication, the CMS automatically creates references to the joint publication on the homepages of the two authors. As the crawl may visit one of these homepages before such an update and the other homepage after the update, the archive may end up with inconsistent page crawls. Hence, an archivist may never be sure if the contents collected so far are still consistent with those contents she needs to retrieve next. To make the matter worse, the crawling of a site should be polite with substantial pauses in between subsequent HTTP-requests in order to avoid unduly high load on the site's HTTP-server [12]. As a consequence, capturing a large Web site may span hours or even days, and changes during this time period and temporary unavailability are the norm.

IWAW09, September 30 - October 1, 2009, Corfu, Greece.

"Coherence" is defined by the Oxford English Dictionary[1] as "the action or fact of cleaving or sticking together", or as a "harmonious connexion of the several parts, so that the whole 'hangs together'". Consequently, a coherence defect exists, if some element violates this condition. In case of Web archiving coherence has a temporal dimension: contents are considered to be coherent if they appear to be "as of" time point x or interval $[x; y]$. What appears to be a simple requirement, develops to be complex and almost impossible to achieve: since publishers cannot provide instantaneous copy of the whole Web site, are autonomous, and do not collaborate with each other, limited guarantees can be given about the quality of the crawls.

Figure 1 depicts coherence defects in a Web archive. In this case, the coherence defects are caused by references from a Web page to other pages, which have already been superseded by more recent versions. In this case, the archived documents on the left-hand side are incoherent (highlighted by a red frame) with respect to the entry page (reference time point "as of" 17/02/2007). However, the link from the entry page to the page on the right-hand side that has been archived on 19/02/2007 is coherent (indicated by a green frame), because both pages are valid and unchanged "as of" 17/02/2007. What is easy to recognize to be incoherent as a human is more difficult for a machine. A computer might only be able to a (very) limited degree interpret the temporal aspect of a page. Nevertheless, given the last modification dates as reference time points of observation, we are able to reason about coherence defects between two instances of

[1]`http://dictionary.oed.com`

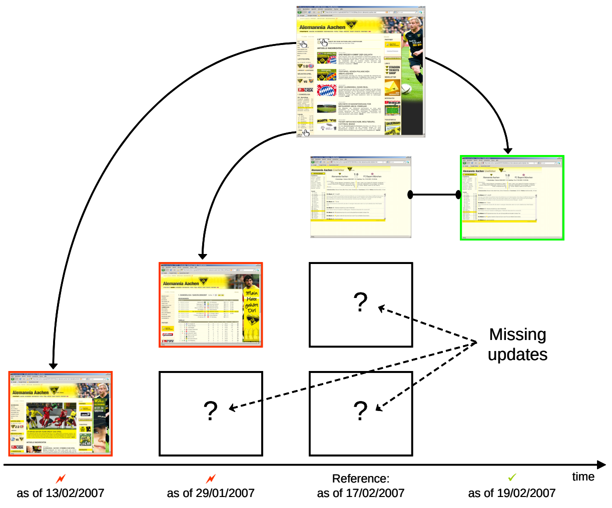


Figure 1: Coherence defects in a Web archive for `www.alemannia-aachen.de` "as of" 17/02/2007

a document. To this end, we introduce several techniques at different levels of granularity to reason about the time point when contents have been modified and – subsequently – analyze if coherence defects exist.

Our research on the visual analysis of coherence defects in Web archiving aims at specifying the degree of change having occurred either during a site crawl or among a sequence of crawls of this site. Thus, the fidelity and interpretability of the crawl becomes quantifiable. Like chasing an impostor, we may not be able to prevent Web contents from changing its appearance, but we are able to identify these changes and to adjust crawling strategies so that future crawl will be as coherent as possible. To better understand the coherence defect and changes in the Web archives we suggest to analyze the data using four different visualization templates: (i) visualization of changes in the spanning tree of the crawl with visone, (ii) scatterplot visualizations of content change analysis, (iii) area plot visualizations of time series, and (iv) scatterplot visualization of time series of change. The visualizations allow investigating coherence defects in terms of both structure and content, get the level of change, and understand the patterns of change among individual pages in a sequence of Web archives. UKGOV and MPI dataset were primarily used in our visual analysis. The UKGOV dataset consists of 120 weekly crawls of seven governmental sites in the united kingdom. MPI dataset consists of daily crawls of the Max Planck Institute for Informatics Web site.

The paper is organized in the following way. We review related work in Section 2. Section 3 introduces change detection and the concept of coherence defects. Section 4 discusses data collection, extraction, and preparation issues for the analysis of coherence defects. Section 5 presents four visualization templates and visually analyzes coherence defects. Finally, Section 6 draws conclusions and gives an outlook on future work.

2. RELATED WORK

The most comprehensive overview on Web archiving is given by Masanès [12]. This book covers the various aspects involved in the archiving as well as the subsequent accessing process. The issue of coherence defects is introduced as well, but only some heuristics how to measure and improve the archiving crawler's front line are suggested. In [19] we have studied the aspect of data quality in Web archiving. Here, we investigated the concept of temporal coherence in more detail, but have not investigated the nature of coherence defects in real world data. In another study, we have designed strategies to minimize the number of changes encountered for random queries in a predefined observation interval [7]. In both studies, we assume that the changes of the pages are well understood and structured. In this paper we do not make any assumptions about the change model. Here, we aim at visually investigating the changes patterns and the coherence defects they cause.

Related research that comes closest to the analysis of coherence defects deals, for instance, with the visualization of web site reconstructions. In this study, McCown et al. evaluate crawling policies which can be used to reconstruct websites from different resources (such as the Internet Archive or search engines) [13, 14]. Their results show which content types might be recovered best and how to reconstruct a Web site as best as possible. The authors also analyze the differences between original and reconstructed Web sites. However, they do not recommend an automated evaluation strategy apart from shingling for text-based resources. In a different direction goes the work of Jatowt et al. [11]. Here, the authors aim at detecting the age of page content in order to dynamically reconstruct page histories at userŠs request. Similarly, Nunes et al. strive for dating Web documents by analyzing their neighbors [17]. Both studies, complement traditional approaches based on relying on HTTP headers or content metadata only. However, both approaches do not analyze the impacts of the measured changes on the overall site's coherence and the visual interpretation of those defects. Other related research mostly focuses on aligning crawlers towards more efficient and fresher Web indexes. Brewington and Cybenko [1] analyze changes of Web sites and draw conclusions about how often they must be reindexed. The issue of crawl efficiency is addressed by Cho et al. [5]. They state that the design of a good crawler is important for many reasons (e.g. ordering and frequency of URLs to be visited) and present an algorithm that obtains more relevant pages (according to their definition) first. In a subsequent study Cho and Garcia-Molina describe the development of an effective incremental crawler [2]. They aim at improving the collection's freshness by bringing in new pages in a timelier manner. Into the same direction head their studies on effective page refresh policies for Web crawlers [3]. Here, they introduce a Poisson process based change model of data sources. In another study, they estimated the frequency of change of online data [4]. For that purpose, they developed several frequency estimators in order to improve Web crawlers and Web caches. In a similar direction goes research of Olston and Pandey [18] who propose a recrawl schedule based on information longevity in order to achieve good freshness. Ipeirotis et al. [10] apply survival analysis to investigate the information longetivity. They devise update schedules based on Cox's proportional hazards regression. Tan et al. [20] use sampling to detect and predict page updates. They identify features reflecting the link structure, the directory structure and the Web page content. Their adaptive download strategies are based on the detected page clusters. Another study about crawling strategies is presented by Najork and Wiener [16]. They have found out that breadth-first search downloads hot pages first, but also that the average quality of the pages decreases over time. Therefore, they suggest performing strict breadth-first search in order to enhance the likeliness to retrieve important pages first. Analysis and understanding of coherence defects is quite different and more difficult. We visualize the changes and coherence defects appropriately and aim to identify pages and subgraphs of the Web crawls that need adjustment in future crawls.

3. CHANGE DETECTION AND COHERENCE DEFECTS

Coherence is a data quality characteristic. In general settings this means that a set of data items do not conflict in terms of predefined constraints. In central database systems the transaction management subsystem ensures that data quality constraints are satisfied. In a distributed environment the individual components must cooperate and use specialized algorithms to ensure the constraints.

Coherence is a complicated matter in Web archiving. The content producers (authors) may post information that is conflicting (a Web page of a soccer match may point to pictures of a different soccer match), the content providers (Web sites) have limited ability and desire to cooperate, and – finally – logics in CMSs are different (pages can be updated immediately, while others may delay the changes; contents may be generated dynamically, depending on download time, location, and Web server load).

In this paper we approach coherence defects from a temporal perspective. It takes an interval of time to archive the whole Web site, however if the archive consists of the versions of pages that can be viewed to be downloaded at one point in time then we say that the archive is without coherence defects. Alternatively, if one of the pages changed during the crawl then there is no guarantee that all the pages can be viewed as of some time point and we say that there is a coherence defect. In order to reason about coherence defects between any two content instances, we employ both the dating and the content check. The canonical way for time stamping a Web page is to use its Last-Modified HTTP header, which is unfortunately unreliable (cf. [6, 11] for more details). For that reason, another dating technique is to exploit the content's semantic timestamps. This might be a global timestamp (for instance, a date preceded by "Last modified:" in the footer of a Web page) or a set of timestamps for individual items in the page, such as news stories, blog posts, comments, etc. However, the extraction of semantic timestamps requires the application of heuristics, which imply a certain level of uncertainty. Finally, the most costly - but 100% reliable - method is to compare a page with its previously downloaded version. Due to cost and efficiency reasons we pursue a potentially multistage change measurement procedure:

1) Check HTTP timestamp. If it is present and is trustworthy, stop here.
2) Check content timestamp. If it is present and is trustworthy, stop here.
3) Compare a hash of the page with previously downloaded hash.
4) Elimination of non-significant differences (ads, fortunes, request timestamp):
    a) only hash text content, or "useful" text content
    b) compare distribution of n-grams (shingling)
    c) compute edit distance from the previous version

On the basis of these dating technologies we are able to develop coherence improving capturing strategies that allow us to reconcile temporal information across multiple crawls and/or multiple archives.

4. DATA EXTRACTION AND PREPARATION

This section discusses data collection, extraction, and preparation issues for the analysis of coherence defects and understanding of change in Web archives. Different visualization templates and coherence defect analyzes put different requirements for the input data. In the simplest case an analysis may require a single instance of a site's archived pages, while more elaborated analysis may investigate the dynamics of change for a pair or even a sequence of instances 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 give a guidance how a database schema should look like (Section 4.1), how to import as well as to clean the data with standard SQL (Section 4.2).

4.1 Database Schema

Essentially, the database schema consists of the pages relationship (cf. t_pages in Figure 10) and links relations (cf. t_links in Figure 10). The pages relationship 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 (cf. t_urls) 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 4.2). The payload of the page is stored in the content attribute provided it did change compared to the previous crawl (cf. vs_page_id, Section 4.3). 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.

Two multidimensional b-trees over attributes (crawl_id, site_id, url_id) for t_pages and (crawl_id, from_site_id, to_site_id, from_url_id, to_url_id, visited_timestamp) for t_links, as well as single attribute b-tree for primary keys exist. This simple but very efficient organization of data allows very fast retrieval of selections (for a given crawl and site ids) and computation of coherence defects (cf. Section 5).

4.2 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 [8] and its successor WARC [9] are a de-facto standard in Web archiving. They are used to record the archived pages themselves and meta data about them (e.g. URL, length, download time, and the checksum of the Web page). 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 [15] 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 need to be cleaned prior to coherence defect 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. Even more, pages can be downloaded multiple times because of the pure formatting of the URLs of the pages: if the Web server does not distinguish the upper and the lower case in the URLs, the designers of the individual pages tend to use different capitalizations of both the filenames and the subdirectories. Large parts of the site can be re-crawled multiple times because of 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 2. 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. 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. Appendix Lines 1–8 in Listing 3), and compute the smallest values for each group (cf. Lines 10–33).

4.3 Data Harvesting with Heritrix

Data harvesting with Heritrix aims at obtaining the data to be stored in the database directly from the crawler, instead of having to do the time-consuming WARC file processing. Even more, information such as the path to a certain page can be directly extracted from the crawler, but needs a complex reconstruction from WARC files. Additionally, we have developed a crawl-revisit mechanism in order to minimize the time frame for the coherence analysis.

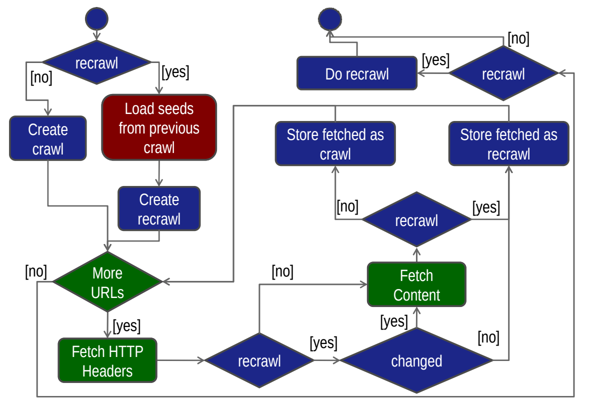


Figure 2: Flowchart of our temporal coherence processor in Heritrix

Technically, our temporal coherence module is subdivided into a modified version of the Heritrix crawler (cf. `crawler.archive.org`), its associated relational database (see Section 4.1 for details about the database design) and an analysis and visualization environment. Into the database, (meta-)data extracted from the modified Heritrix crawler are stored. Furthermore, a crawl-recrawl mechanism has been added, which performs an efficient recrawl strategy. It allows testing for content changes right after the crawl has completed. To this end, we apply conditional GETs that make use of the contents' ETags. As a result, the subsequent validation phase becomes faster by simultaneously reducing bandwidth as well as server load. All crawls are then made accessible as distinct crawl-recrawl pairs. Of course, arbitrary crawls can be combined as artificial crawl-recrawl pairs of "virtually" decelerated crawls as well. Figure 2 depicts a flowchart highlighting the main aspects of our temporal coherence processor in Heritrix. Those elements in green contain unchanged elements compared with the standard Heritrix crawler. The bluish items represent methods of the existing crawler that have been adapted to our recrawl strategy. Finally, the red unit represents an additional processing step required for initializing the instantaneous recrawl.

The analysis and visualization environment of our temporal coherence module serves as a means to measure the quality of a single crawl-recrawl pair or any pair of two crawls. For that purpose, statistical data per crawl (e.g. number of defects occurred sorted by defect type) is computed from the associated relational database after crawl completion. In this post-processing step, the site is analyzed in a (spanning) tree representation derived from the crawler's path within the site.

```
input  : CrawlTree tree
begin
    collapseNode (tree.root)
    drawNode (tree.root)
end
```

Algorithm 1: processCrawlTree

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 defects, we compress the tree and visualize only its relevant components (cf. Algorithm 1).

```
input : Node node
begin
    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
    forall children of node do
        collapseNode (child)
        if child.collapsing=false then node.collapsing=false
        else node.collapsingSize=child.collapsingSize+1
    end
end
```

Algorithm 2: collapseNode

In the first step, we analyze the contents of the spanning tree and "flag" them according to their status: green if they are unchanged (coherent), yellow in case of textual incoherence only, red in terms of structural incoherence observed, and, finally, black whenever a content is missing in the subsequent crawl. Afterwards, we apply a collapsing strategy (cf. Algorithm 2), which tags the nodes in each fully coherent subtree as collapsible and marks the defect nodes and their ancestors as non-collapsible. In the visualization step (cf. Algorithm 3) 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.

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

Algorithm 3: drawNode

For a graphical representation, the previously computed compacted tree representation is exported into a graphML-file (cf. Listing 1). The graphML file format is an XML-based standard and a file format for graphs. It is capable of describing all previously computed structural properties and applied in many graph related software applications.

```
<?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 1: Coherence defect graphML-file (excerpt)

5. COHERENCE DEFECT ANALYSIS

The coherence defect analysis measures the quality of crawls either from crawl-recrawl pairs, between any two crawls, or a series of crawls. To this end, we have developed methods for automatically generating sophisticated statistics and visualizations (e.g. number of defects occurred sorted by defect type).

5.1 Structural and Content Change Analysis with visone

As described Section 4.3, our extension to Heritrix allows us to trace the crawling process with statistical data and export this data compliant to graphML. By applying graphML compliant software it is also possible to layout a crawl's spanning tree and visualize its coherence defects. This visual metaphor is intended as an additional means to automated statistics for understanding the problems that occurred during capturing. Its main field of application is the analysis of high quality (single) Web site crawls.

Figure 3 depicts a sample visualization of an `mpi-inf.mpg.de` domain crawl (about 65.000 Web contents) with the visone software (cf. `http://visone.info/` for details). Depending on the nodes' size, shape, and color the user gets an immediate overview on the success or failure of the capturing process. In particular, a node's size is proportional to the amount of coherent Web contents contained in its sub-tree. In the same sense, a node's color highlights its "coherence status". While green stands for coherence, the signal colors yellow and red indicated (content incoherence and/or link structure

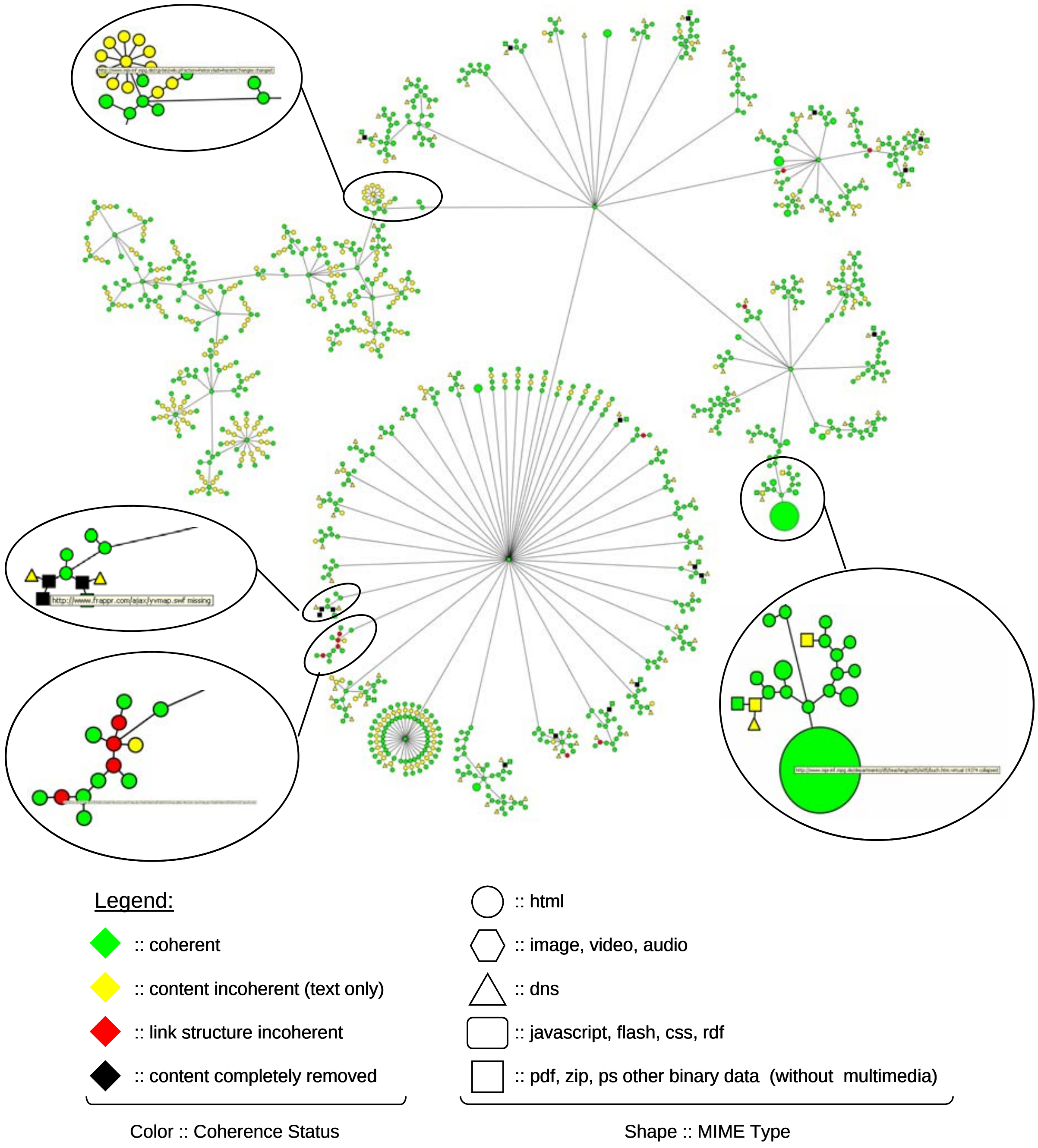
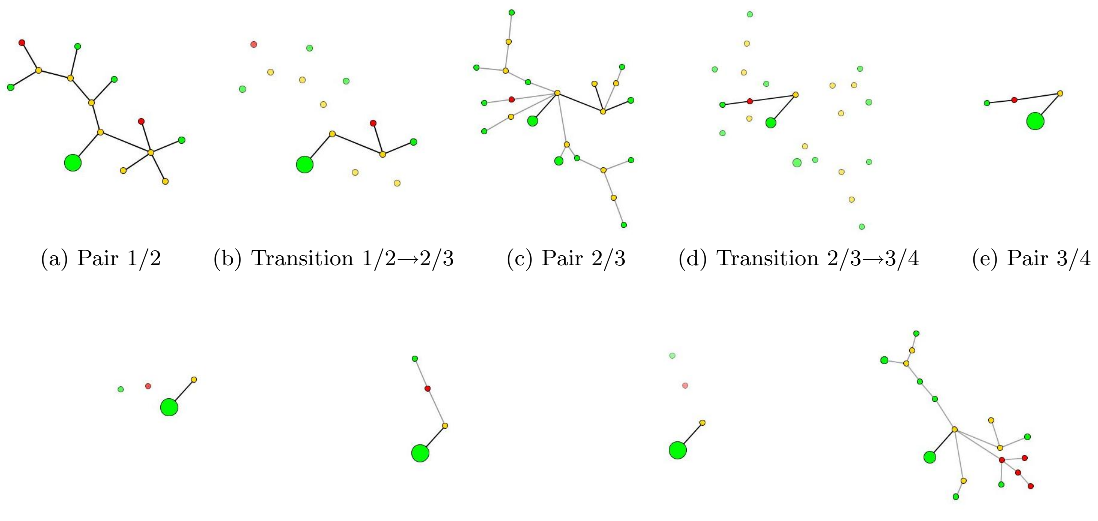

http://www.frappr.com/ajax/yvmap.swf missing
Legend:
🟩 :: coherent
🟨 :: content incoherent (text only)
🟥 :: link structure incoherent
⬛ :: content completely removed
Color :: Coherence Status
○ :: html
⬡ :: image, video, audio
△ :: dns
▢ :: javascript, flash, css, rdf
□ :: pdf, zip, ps other binary data (without multimedia)
Shape :: MIME Type

Figure 3: Coherence defect visualization of a single crawl-recrawl pair of `mpi-inf.mpg.de` by visone


(a) Pair 1/2 (b) Transition 1/2→2/3 (c) Pair 2/3 (d) Transition 2/3→3/4 (e) Pair 3/4

(f) Transition 3/4→4/5 (g) Pair 4/5 (h) Transition 4/5→5/6 (i) Pair 5/6

Figure 4: Tracing of coherence defects in crawl-recrawl pairs of the `dmoz.org/news` subdomain over time

incoherence). The most serious defect class of missing contents is colored in black. Finally, a node's shape indicates its MIME type ranging from circles (HTML contents), hexagons (multimedia contents), rounded rectangles (Flash or similar), squares (PDF contents and other binaries) to triangles (DNS lookups).

An extension to a single crawl-recrawl pair analysis is their analysis over time. The idea behind this approach is to trace coherence defects among several crawls and to identify those contents, which are less frequently changing and – thus – more likely to be coherent. Figure 4 depicts a coherence defect visualization series of six subsequent crawls on the `dmoz.org/news` subdomain. For each subsequent pair of crawls a coherence defect visualization like in the previous example is being performed. In contrast to before, now we compare the crawls themselves instead of their crawl-recrawl pairs. In the transitions between any two of these pairs all those nodes are faded out, which disappear in the coming coherence defect analysis. In contrast, those contents showing the same coherence characteristic between two crawl pairs remain solid and the newly appearing nodes are located around them. Interesting to see in this example is that there is a solid core of a large coherent subtree and its always content-wise incoherent predecessor node.

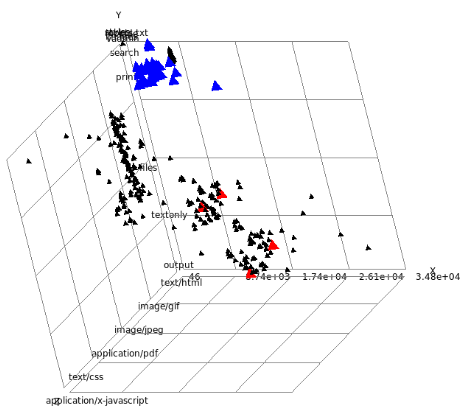
5.2 Content Change Analysis with Scatterplots

Two and three-dimensional scatterplots can be employed to visualize, locate and analyze the content changes and to some extent the structural changes. Figures 5–6 show scatterplot visualizations for the `sabre` and `royal-navy` sites from the UKGOV database. Here the scatterplot visualizations map the mime-type, the size 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 blue, changed pages are colored red, while the black color depicts unchanged pages). The archivist should look for patterns of changed and added pages. For example from Figure 5(a) one can see that there are a few changes in the HTML files in the output and textonly subdirectories (cf. the red points in the figure) and a whole new subdirectory of files are added into the Web archive (cf. the blue points at the top of the visualization). The changes of the pages (cf. the four red 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 Crawl 2 indicating a high quality of the archive.

Likewise, patterns of newly added images are similar for the `royal-navy site` (cf. image/jpeg and image/gif pages in Figures 6(a)–6(b)), while the patterns of newly added and changed HTML pages differ slightly showing the changing structure of the Web site.

5.3 Time Series Analysis with Area Plots

The series of the area plots (cf. Figures 7–8) can be used to get an overview of the percentage of change in the crawls of Web archives of sites as the crawl id increases. The X axis maps to the crawl id, while the Y axis maps either to the number of changed/unchanged/new/deleted pages or the percentage/-download time. Separate figures are typically drawn for links (graph structure of the site, cf. Figures 7(a)–8(a)), HTML pages (cf. Figures 7(c)–8(c)), images (cf. Figures 7(d)–8(d)) (content changes) along with a figure for a similarity and the time of download (cf. Figures 7(b)–8(b)). The archivist should look for patterns that significantly change at certain time points. For example, one can see that there is a significant change in the Web site in crawls 52 and 93. The crawl time suggests that in crawl 52 the Web site underwent significant changes, however in crawl 93 the quality of the


(a) Crawl 1

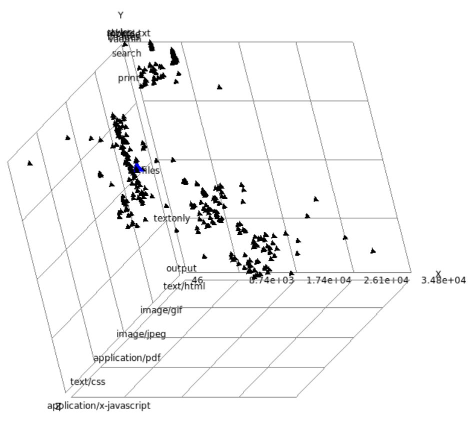
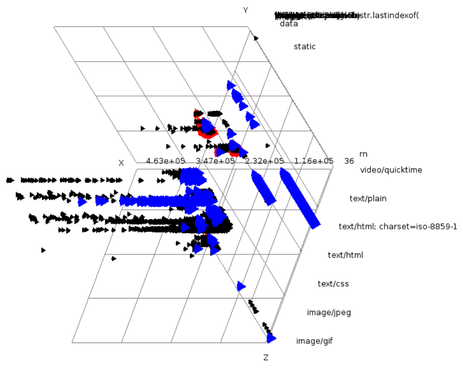

(b) Crawl 2

Figure 5: Crawl sequence analysis of `www.sabre.mod.uk` site


(a) Crawl 1

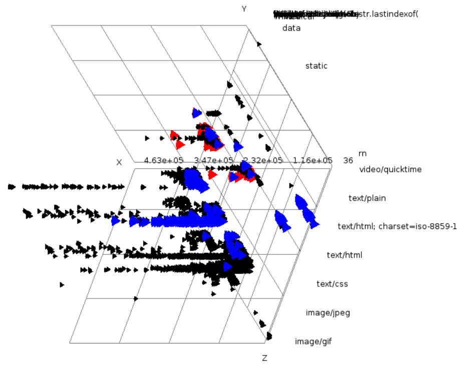
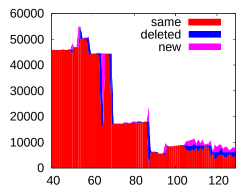
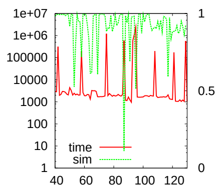
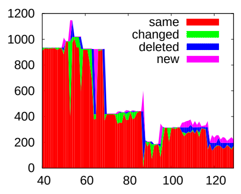

(b) Crawl 2

Figure 6: Scatterplot analysis of `www.royal-navy.mod.uk` site


(a) Links


(b) Similarity and Time


(c) HTML pages

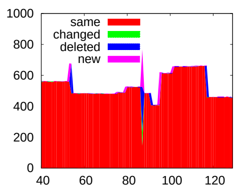
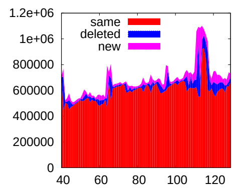
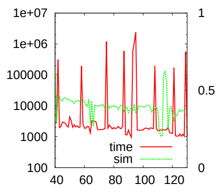
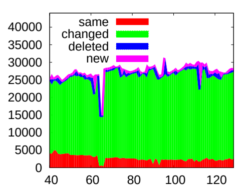

(d) Images

Figure 7: Crawl sequence analysis of `www.sabre.mod.uk` site


(a) Links


(b) Similarity and Time


(c) HTML pages

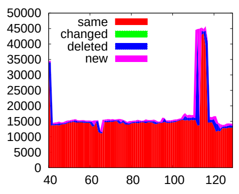

(d) Images

Figure 8: Crawl sequence analysis of `www.royal-navy.mod.uk` site

archive decreased due to some archiving issues. Even more, one can see that the Web site itself greatly shrank over time.

Computation of area graphs can be expressed in pure SQL and optimized by the query optimizer (cf. Listing 4 in the Appendix) resulting in the overall $O(n \log n)$ or better complexity. The algorithm selects the tuples of the site of crawl XXX and the previous crawl (cf. Lines 31–40) and uses the full outer join to join the crawls. The tuples that are in one or the other crawl (but not both) are new and deleted pages, while the remaining tuples result into either changed or the unchanged ones (cf. Lines 10–15). The new, deleted, changed, and unchanged tuples are grouped and summed up (cf. Lines 1-10).

5.4 Time Series Analysis with Scatterplots

Change patterns in the pages of one site can be analyzed with a scatterplot of time series data (cf. Figure 9). In the figure the Y axis maps the pages, the X axis maps the crawl id, and a cross is visualized at (crawl id, page) position if there was a change in the Web page at the given crawl id compared to the previous crawl id. The pages on the Y axis are ordered so that pages showing a similar change behavior are placed close to each other.

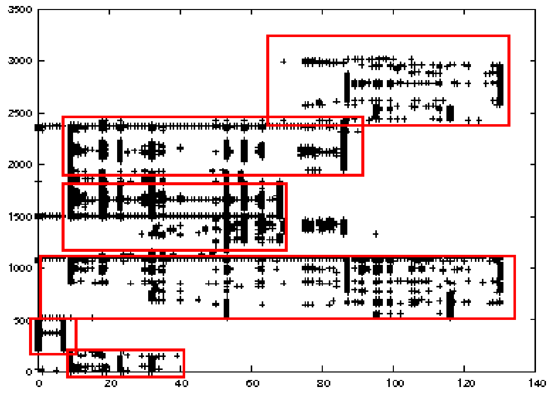


Figure 9: Scatterplot of lines for `www.sabre.mod.uk`

The figure helps the Web archivist to detect and analyze pages that have similar change and coherence defect patterns. The visualization clearly separates the pages of the Web site into distinct blocks (cf. rectangles in the visualization) identifying different patterns of change and coherence defects.

6. LESSONS LEARNED AND FUTURE WORK

From an archiving point of view, the ideal case in Web archiving would be to prevent contents from changing during crawl time. Of course, this is illusion and practically infeasible. Consequently, one may never be sure if the contents collected so far are still consistent with those contents to be crawled next. However, temporal coherence in Web archiving is a key issue in order to crawl digital contents in a reproducible and, thus, later on interpretable manner. To this end, we have developed an extension of Heritrix that is capable of dealing with proper as well as improper dated contents. Hence, we are able to make coherence defect analysis more efficient, regardless of how unreliable Web servers are. Altogether, the analysis and visualization features developed aim at helping the crawl engineer to better understand the nature of coherence defects within or between Web sites and – consequently – to adapt the crawling strategy/frequency for future crawls. As a result, this will also help increase the overall archive's coherence.

While our coherence defect analysis now helps us to understand incoherence more systematically, future research needs to make these insights productive. Hence, ongoing research aims toward partial crawling and increased archive coverage (e.g. more time points / complete interval). Even more, the combination of partial recrawls in combination with an increased (partial) crawling frequency might be particularly appealing from an efficiency point of view. In addition, results obtained from real life crawl analysis will be useful for the creation of sophisticated simulation environments. Hence, we will be able to resemble real life change behavior and simulate real life Web site topologies in a simulation framework.

Acknowledgements

This work was supported by the 7^{th} Framework IST programme of the EC through the small or medium-scale focused research project (STREP) on Living Web Archives (LiWA) contract no. 216267. We thank our colleagues for the inspiring discussions.

7. REFERENCES

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

[2] Junghoo Cho and Hector Garcia-Molina. The evolution of the web and implications for an incremental crawler. In *VLDB '00: Proceedings of the 26th International Conference on Very Large Data Bases*, pages 200–209, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[3] Junghoo Cho and Hector Garcia-Molina. Effective page refresh policies for web crawlers. *ACM Transactions on Database Systems*, 28(4), 2003.

[4] Junghoo Cho and Hector Garcia-Molina. Estimating frequency of change. *ACM Trans. Inter. Tech.*, 3(3):256–290, August 2003.

[5] Junghoo Cho, Hector Garcia-Molina, and Lawrence Page. Efficient crawling through url ordering. In *WWW7: Proceedings of the seventh international conference on World Wide Web 7*, pages 161–172, Amsterdam, The Netherlands, The Netherlands, 1998. Elsevier Science Publishers B. V.

[6] L. Clausen. Concerning etags and datestamps. In A. Rauber J. Masanès, editor, *4th International Web Archiving Workshop (IWAW'04)*, 2004.

[7] Dimitar Denev, Arturas Mazeika, Marc Spaniol, and Gerhard Weikum. Sharc: Framework for qualityconscious web archiving. In *VLDB '09: Proceedings of the 35th international conference on Very Large Data Bases.* VLDB Endowment, 2009.

[8] International Internet Preservation Consortium. Arc_ia, internet archive arc file format. `http://www.digitalpreservation.gov/formats/fdd/fdd000235.shtml`.

[9] International Internet Preservation Consortium. Warc, web archive file format. `http://www.digitalpreservation.gov/formats/fdd/fdd000236.shtml`.

[10] Panagiotis G. Ipeirotis, Alexandros Ntoulas, Junghoo Cho, and Luis Gravano. Modeling and managing changes in text databases. *ACM Trans. Database Syst.*, 32(3):14, 2007.

[11] Adam Jatowt, Yukiko Kawai, and Katsumi Tanaka. Detecting age of page content. In *WIDM*, pages 137–144, 2007.

[12] Julien Masanès. *Web Archiving*. Springer, New York, Inc., Secaucus, NJ, 2006.

[13] Frank McCown and Michael L. Nelson. Evaluation of crawling policies for a web-repository crawler. In *Hypertext*, pages 157–168, 2006.

[14] Frank McCown, Joan A. Smith, and Michael L. Nelson. Lazy preservation: reconstructing websites by crawling the crawlers. In *WIDM*, pages 67–74, 2006.

[15] G. Mohr, M. Kimpton, M. Stack, and I. Ranitovic. Introduction to heritrix, an archival quality web crawler. In *4th International Web Archiving Workshop (IWAW'04)*, 2004.

[16] Marc Najork and Janet L. Wiener. Breadth-first search crawling yields high-quality pages. In *In Proc. 10th International World Wide Web Conference*, pages 114–118, 2001.

[17] Sérgio Nunes, Cristina Ribeiro, and Gabriel David. Using neighbors to date web documents. In *WIDM*, pages 129–136, 2007.

[18] Christopher Olston and Sandeep Pandey. Recrawl scheduling based on information longevity. In *WWW '08: Proceeding of the 17th international conference on World Wide Web*, pages 437–446. ACM, 2008.

[19] M. Spaniol, D. Denev, A. Mazeika, P. Senellart, and G. Weikum. Data Quality in Web Archiving. In *Proceedings of WICOW, Madrid, Spain, April 20, 2009*, pages 19 – 26. ACM Press, 2009.

[20] Qingzhao Tan, Ziming Zhuang, Prasenjit Mitra, and C. Lee Giles. Efficiently detecting webpage updates using samples. In *ICWE*, pages 285–300, 2007.

APPENDIX

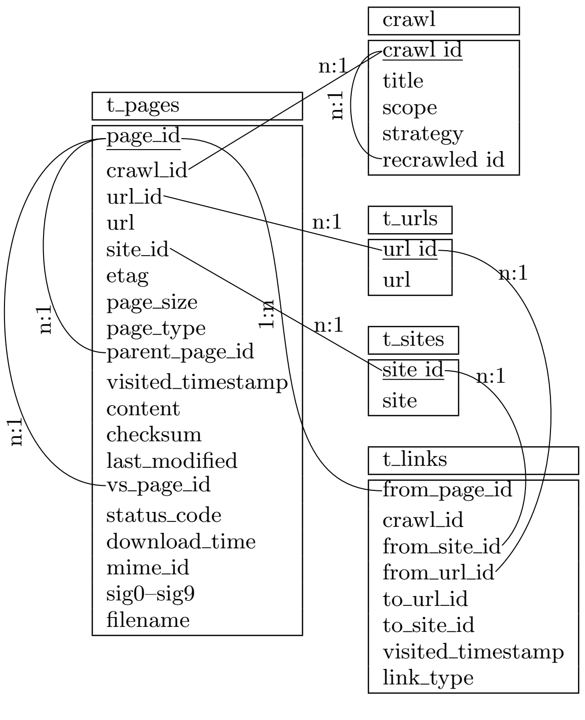


Figure 10: DB Schema

```
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 2: 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 3: SQL for cleaning to lower urls

```sql
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
         from nodes
         where crawl_id = XXX
       ) as ncurr
       on nprev.site_id = ncurr.site_id and
       nprev.url_id = ncurr.url_id
     ) as crawls_with_status
     group by ssite_id, status
   ) as crosstab_multiple_rows
  ) as crosstab_single_row
  group by site_id
) as aggregated
```

Listing 4: SQL for area bars