ArcSpread for Analyzing Web Archives

Siddhi Soman, Arti Chhajta, Alexander Bonomo*, Andreas Paepcke

Stanford University
353 Serra Mall
Stanford, CA 94305
ssoman@stanford.edu,
achhajta@stanford.edu
paepcke@cs.stanford.edu

University of California
4732 Boelter Hall
Los Angeles, CA 90095
abonomo@ucla.edu

ABSTRACT
We describe an architecture, partial implementation, and user study for ArcSpread. The vision for ArcSpread is to allow social scientists of the future, such as Historians, or Political Scientists, to analyze Web archives through a spreadsheet-like interface. Cells of these spreadsheets contain sets of objects, rather than single items. Examples for objects are Web page, Image, and Word. Formulas perform set operations on cell contents. When new content must be acquired, the formulas access an SQLite cache, or trigger operations on an underlying 60-core Hadoop cluster. This cluster, and the spreadsheet formulas have access to a multi-terabyte Web archive. When users double click on a loaded cell, a browser appropriate for the cell content type is raised. We present an envisioned example interaction, sketch the implemented Hadoop and spreadsheet level facilities, and describe the prototype Word Browser, Web Page Summarizer, and Page Classifier. We conducted a user study to evaluate the quality of the summarizer.

ACM Classification: H.5.2 [Information Interfaces and Presentation]: Graphical user interfaces (GUI). H.5.4 [Information Interfaces and Presentation]: Architectures.

General terms: Design, Documentation

Keywords: Web archive, historian, social science, web science

Introduction
As the World Wide Web has developed into the primary carrier of several societies’ cultural, political, scientific, and commercial proceedings, Web archives have emerged that intend to capture this evolving history. Future historians will need to explore these archives, as they now explore paper based primary sources. Political Scientists and Sociologists must even now delve into (more recent) layers of these archives for retrospectives on election campaigns, or the evolution of controversy. For example, Web ‘chatter’ immediately after the U.S. Katrina Hurricane disaster revolved around water levels and dam structures. At some point discussion changed to one of race. The early origins of this pivot point are buried in Web archives. Finding their trace using currently available Web archive tools would be cumbersome.

Other examples for future roles of Web archives are research into questions like “how have successive government administrations used the Web as a tool?”, and “how has the English language changed over time?”

The Web’s size has necessitated a division of labor between archival institutions, as has occurred for archives with paper content. Some institutions are breadth oriented, like the Internet Archive [3]. Others, like the U.S. Library of Congress, focus on particular events or topics [24]. Many nations are evolving archives to hold Web materials created within their jurisdiction.

At a smaller granularity, college libraries, and even individuals can archive materials from their area of interest through facilities like the Internet Archive’s Archive-It service [2].

Aside from policy restrictions (many European national archives limit access to terminals in a central building), three access models prevail for these archives. The most common is keyword search, identical to what is familiar from search engine interactions.

A second model, offered most prominently by the Wayback Machine [4], is access based on URL and time. Users enter the URL of a Web site, and a time \( t \) in the past. The Wayback Machine then displays the Web page as it was at time \( t \) at that URL.

The final model is ‘wholesale’ access, where entire archive portions may be streamed to a client for local analysis. The Stanford WebBase is an example for this approach [15, 8].

All three models have their use. For example, the Wayback Machine access model is popular with patent lawyers looking for prior availability of information pertinent to a patent. The wholesale model has been of service to computer scientists, for example for analysis of Web graph structures.
None of the three access modalities are matches for the needs of someone trying to ‘make sense’ of an archive collection. That activity requires, among other capabilities, comparisons between Web content of different time periods, or the evolution of links on those pages. For instance, it was observed by coincidence that outgoing links to reproductive health related studies present during the Clinton U.S. administration disappeared once his successor gained control over the government sites.

ArcSpread is a step towards enabling more exploratory interaction with Web archives than is afforded by any of the three described access modes. Another goal is to provide tools needed for rigorous analysis of the results. While the precise nature of future needs in this area is largely unexplored, trends in the area of Digital Humanities are indicators. Efforts in that area have been evolving from earlier digitization projects [10] to the analysis of digital artifacts via visualizations and computation [31].

However, even if these trends are beginning to reveal future needs, unforeseen details emerge in practice. For example, [29] report that their work on near-duplicate detection was motivated by the mid-stream realization of collaborating Political Scientists that Web pages from different sites, with nothing but a small Associated Press article in common were for their purposes just duplicates of each other, even though the pages otherwise looked radically different. Figure 1 summarizes analysis needs identified from conversations with Political Scientists and Historians.

ArcSpread’s intellectual and architectural design is a framework that both inspires and supports semi-independent research whose prototypes then integrate at well defined points with other such prototypes. The purpose of this paper is both to explain ArcSpread’s concepts, and to demonstrate its dual purpose of generating and integrating diverse research. The overall goal is to boost the usefulness of Web archives for purposes present and future.

We begin with an ArcSpread interaction example as we envision the final result. Following a subsequent overview of our software architecture, we describe elements of our existing vertical implementation slice through that architecture. We finish with related work and a conclusion.

**Envisioned Interaction**

We walk through a complete example, using a mock-up. The individual components of this mock-up are implemented, but they have not yet been integrated (see implementation section).

Consider the following analysis goal. For a Web archive’s crawl1 on March, 2011 show:

- Groups of images on pages that contain, in turn: ‘debt,’ ‘infrastructure,’ ‘war.’
- People and place names on those pages.
- The most frequent of those names.
- Web pages containing the most frequent name in addition to ‘debt,’ ‘infrastructure,’ or ‘war.’

Then change the entire analysis to cover a different crawl.

Figure 2 shows the final envisioned result of this activity.

Cell B1 conceptually contains all the pages in the crawl called gen-03-2011. Row 5 contains a series of results for pages that contain the word ‘debt,’ row 6 holds results for the pages that contain the word ‘infrastructure,’ and so on.

The cells in column B hold sets of Web pages. For example, cell B5 contains the pages that contain ‘debt.’ Column C contains the images of the pages in column B. Place and person names from these pages are collected in column D, E and F respectively contain the most frequent person name, and pages containing that name in addition to the term in column A of their row.

The following figures demonstrate how the uses of set enriched spreadsheet formulas work together to accomplish the final sheet of Figure 2.

**Loading archive content.** Figure 3 shows how a formula populates a cell with Web pages from the underlying archive. The string gen-03-2011 identifies the portion of the archive to retrieve. In this example that granularity is one crawl.

The load() function triggers retrieval. An implementation may or may not retrieve some or all of these pages from the archive early.

---

1A crawl is a collection sweep through the Web that an archive undertakes during a particular time period.
Selecting among set members. Figure 4 introduces a piece of syntax: square brackets indicate selection. The expression \( B51[\text{contains}(A$5)] \) translates to 'examine the Web pages in cell B1, and select those that contain the word contained in cell A5'. Cell A5 contains the word 'debt,' one of the words in the problem statement. B5 thus fills with Web pages containing that keyword.

The dollar signs have the same meaning they hold in the Excel application. These modifiers exert their effect when a formula is copied. During such copy operations cell references are normally adjusted such that the cells addressed in the formula retain the same distance from the target of the paste as the originally addressed cells had from the source cell of the copy/paste operation. The dollar symbol prevents this adjustment. Any Excel tutorial can further explain this process. We return to ArcSpread’s copy/paste operation below.

Extracting content from one set member. Figure 5 introduces a second operation, the dot.

This operation extracts one attribute from an object in a set. In the example of the Figure the image attribute, meaning all of each page’s images are extracted from each Web page object. The operation creates a union of all these images. Another Web page attribute example is its URL. In database terms the dot operation is similar to projection.

Applying functions to all set members. As in traditional spreadsheets, functions may be associated with cells. These functions are re-evaluated if necessary, as the content of other cells changes. The example in Figure 6 applies the function properNames() to each Web page in cells B5, B6, and B7, respectively.

This function uses the Stanford Parts-of-Speech tagger to identify names of people, places, and organizations [30]. Other implemented functions are the usual set operations and aggregations. We are adding functions for access to underlying processing facilities, like classification, as they come online (see section on implementation).

Figure 7, which shows the construction of the next column towards the list of goals, illustrates that sometimes functions at the spreadsheet level will wrap idioms of an underlying SQL engine. In this case, the wordCount() function implies a groupBy operation.

Figure 8, finally, shows how the final column is created. For example, column F5 will contain those Web pages in crawl gen-03-2011, which contain the word ‘debt,’ and also the
proper noun ‘Obama,’ because ‘Obama’ is the most common proper noun in the ‘debt’ pages.

**Automatic recomputation.** As in traditional spreadsheets, changing a cell’s contents triggers the examination of a dependency graph. Any cells that reference the modified cell are recomputed. For example, the keywords of column A can be changed, thereby changing content populations of the respective rows. More far reaching is a change in B1. Changing the crawl identifier there will recompute all the remaining cells based on an entirely different section of the underlying archive.

However, rather than making that change, a more informative option is shown in Figure 9.

In the Figure all the formulas in B5:F7 are copied to the cell rectangle with upper left corner at G5. The source crawl specification in G1 is changed—in this example to the U.S. Federal Government crawl of the same time period—and a comparison between the two crawls is available after recomputation. The automatic relocation of cell references that are not decorated with the dollar modifier ensures that the proper selections, projections, and function applications are applied to the second crawl as they were applied to the first one.

**Vertical Implementation Slice**

The work on ArcSpread is partitioned into three layers as shown in Figure 10.

The bottom, or *machine room* layer consists of a 60 core Hadoop cluster. The map-reduce jobs are programmed via Pig [27], a cross between relational and procedural computational views onto Hadoop. The task for this layer is to ingest content streams from Web archives, and to generate specialized results that are then stored in various cache facilities, notably an SQLite database.

Our prototype draws on the Stanford WebBase [15, 8] as its test source archive. We also enabled both our Hadoop cluster and the spreadsheet engine to ingest WARC archive files, a format used by the Internet Archive, the Library of Congress, and others [25].

Jobs are launched by the *Sheet Engine*, which computes the spreadsheet formulas. Our currently implemented machine room facilities for streams of Web content are word index construction, word pair distance extraction, part of speech tagging, word count, link anchor ALT text extraction, and classification. The addition of new facilities involves the creation of Pig scripts.

The *spreadsheet interaction layer* uses an existing engine that was used for PhotoSpread [19], and is publicly available on Github. Basics of the formula language are documented at [18]. The addition of new functions is accomplished by Java class extensions.

Type specific browsers are stand-alone applications that are invoked upon double clicking a populated cell (left edge of Figure 10). These browsers may obtain data from the spreadsheet runtime, directly via Java iterators over an underlying archive, or from the cache database. The PhotoSpread *image workspace* serves as browser for images. We have implemented the beginnings of a Web page summarization browser for cells that contain Web pages, and a word exploration browser for exploring cells with sets of words. We
Figure 10: Three-tiered ArcSpread architecture.

Figure 11: Early prototype of the ArcSpread Word-Browser.

introduce these prototypes in the following two subsections.

**Word Browser**

The WordBrowser is intended to help investigators explore historic Web linguistics, and other language related phenomena, like the juxtaposition of particular adjectives with politicians’ names. While aggregate word statistics and visualizations are essential to the successful interaction with large word sets, we are also experimenting with drill-down to the texts themselves. Figure 11 illustrates this intent.

The right pane of the display holds the text to be examined. The left pane holds a column of *inserts* that implement a filter chain. Filters are additive such that stacking order of the inserts is significant. Each insert is customized via checkboxes, or through a separate preference dialog. The example of Figure 11 shows a named entity resolution filter insert. Depending on which checkbox is activated, Person, Location, or Organization, this insert highlights respective occurrences in the text. Other existing inserts that are yet to be integrated into the user interface highlight word length, frequency, part of speech, or link anchors. The browser is extensible with more specialized future inserts, like idiom detectors, or synonym usage for a given word.

The current implementation saves selected words with the URL of their page in comma-separated value files. The plan is to allow drag-drop of those sets into spreadsheet cells.

Beyond this WordBrowser, we implemented a second example cell browser. This application summarizes individual Web pages.

**Web Page Summarization**

Our Web page summarization takes a two pronged approach. First, we attempt to extract the most ‘important’ text snippets from the page. Second, we try automatically to rank the page’s images by importance. The page summary is then a collage of the resulting snippets and images.

**Text Snippet Extraction** We combine three known measures for our text extraction. The finally selected summary text snippet is a linear combination of these three measures. The appropriate weights for each measure were learned through a support vector machine.

**DOM node tf*idf**. This second text snippet importance measure is based on ‘term frequency*inverse document frequency’ ($tf \times idf$) maximization [17] as follows. We created a reference set of news Web pages. Given an input page $P$, we generate its DOM graph. We then compute an importance score for each of that graph’s nodes $N_i$. We arrive at this score by summing the $tf \times idf$ scores for each word in $N_i$ relative to the reference set. The node with the highest sum receives $P$’s highest DOM node $tf*idf$ measure score.

Note that in actual operation Web page $P$ may or may not be a news page. Depending on the page’s true genre our reference set may be more or less appropriate. Our overall results might therefore be improvable by using several reference sets of different genre.

**Relative noun count.** The final component measure for our summary snippet selection is based on parts of speech analysis. The measure is computed for each DOM node as the count of nouns in that node divided by the total number of words in the node; the higher this relative noun count, the higher the node score.

The final importance score of each DOM text node $T$ is computed as

$$s_{text}(T) = w_1 \times m_1 + w_2 \times m_2 + w_3 \times m_3 + b \quad (1)$$

where $w_i$ is a model weight, and $m_i$ is one of the scores described above. We used a support vector machine (SVM) to determine the weights $w_i$. The publicly available, hand coded gold collection of [29] served as training set.

**Representative Images** In addition to text snippets, we choose ‘important’ images to represent a Web page. We consider
three factors while trying to pick out the important images in a page, and we tested our method in a user study.

In an initial step we remove all images from known advertising internet domains. We then compute the following three scores.

**Image size score.** We hypothesize that absent advertising, larger images will usually be considered more important by a Web page publisher than images that are icons or thumbnail sized. We thus sort the images by size, and compute the following measure as a size score for the size-sorted images \(I_1...I_n\).

\[
s_{\text{size}}(I_j) = 1 - (j/n)
\]

where \(j\) is the position of \(I_j\) in the sort order, and \(n\) is the number of images on the page. A large image will thereby be scored higher by the size measure than a smaller one.

**Image position score.** Our second measure considers an image’s position on its Web page. The intuition behind this score is that the images closer to the beginning of the Web page are more important than images further down, since the publisher wished the user to view the top images first.

To compute an image’s position score we sort the respective Web page’s images by their position in the DOM tree. We then normalize as in equation 2. DOM tree position is obtained by a breadth-first flattening of the tree.

**Text snippet proximity.** Our final constituent image importance measure is an image’s proximity to the \(m\) most important DOM text nodes on the respective page, where we currently set \(m = 4\). We hypothesize that images closer to important text will also be important. Proximity is again determined by distance in the flattened DOM tree. An image \(I\)'s text snippet proximity score is computed as the normalized sum of \(I\)'s distances from the top \(m\) most important text snippets.

\[
s_{\text{prox}}(I) = 1 - (\sum_{i=1}^{m} |\text{pos}(T_i) - \text{pos}(I)|)/n
\]

where the \(\text{pos()}\) function returns the distance of its DOM node argument from the DOM tree root. Quantity \(n\) is the number of image nodes in the DOM tree.

As with text snippet measures, we combine these three image scores to assign a weighted final score to each image. The top \(k\) images are used as an image summary for a given Web page. Our early prototype lays out a collage of text snippets and images with the intent to speed human inspection of Web page collections. Users may control the number \(k\) of snippets and images to use in these summaries.

We next discuss a user study performed to evaluate our image summarization.

**Evaluation of Image Ranking**

We prepared a Web facility that presented 30 Web pages to 30 experiment participants in random order. The pages were selected primarily from Open Directory Project (ODP) categories [23], with some categories added manually. The pages spanned *News, Business, Sports, Science, Health, Art, Suggestions/recommendations, Finance, Education, Government/Official, Entertainment, Political, and History*. The pages contained a total of 654 images.

We asked participants to rank ten images on each successively displayed Web page by importance. The instructions requested that participants consider the importance that the respective page’s publisher intended to impart on the images, that is not the importance the participant ascribed. The intent of this instruction was to simulate a person who wants to make sense of a Web archive’s content, rather than approaching the archive with a particular task.

We did not excise any advertising images from the presented pages, but asked participants to disregard advertising related images. Our experiment page allowed participants to click on any image. In response a ‘rank widget’ was displayed in a side bar. The widget contained a thumbnail of the corresponding image, and its rank. Participants were able to rearrange these widgets, and to delete them, until the widgets’ order matched the participants’ image importance rankings.

Participants ranged in age from 18 to 69, with 23 participants in the 18–25 span. Eighteen participants were male, ten were female. The remaining participants did not fill in their gender.

Once all rankings were obtained, we realized that some participants had misunderstood the instructions. We identified the resulting outliers as follows. We computed the bivariate Spearman’s \(\rho\) between all pairs of rankers. We then determined the distribution of non-significant values of \(\rho\), and identified two outliers whose count of such misalignment with the other rankers exceeded twice the standard deviation of other rankers’ disagreement. This procedure eliminated two rankers. Three more rankers were eliminated because of excessive missing values, leaving 25 rankers for the remaining analysis.

These 25 rankers agreed in their rankings at a Krippendorff’s \(\alpha\) value of .647. This level of agreement is not very high, so we followed two procedures in comparing our image importance ranking algorithm to these human rankers. For the first procedure we computed the average rank that each image received from all rankers. We then computed the Spearman’s \(\rho\) correlation between the resulting two rank sets, with pairwise exclusion of missing values. The result of .390 (two-tailed) was significant at \(p < 0.01\).

In consideration of the low Krippendorff’s \(\alpha\) our second procedure answered the softer question of whether our image importance algorithm is distinguishable from the human rankers. We computed the pairwise bivariate Spearman’s \(\rho\) between the algorithm’s rankings and each of the human rankings (pairwise exclusion of missing values). We then determined the number of rankers with whom the algorithm did not produce a significant, or highly significant \(\rho\). We found that the algorithm’s correlation with 19 rankers was highly significant \((p < 0.01)\). The correlation with a 20th ranker was significant \((p < 0.05)\), and five ranker/algorithm pairs were not correlated at a significant level.
Reviewing these results we conclude that our image importance ranking algorithm does well in a very harsh environment. Note that rankers were in general confronted with many more than 10 images on each Web page. This degree of freedom leads to missing values that would likely be available if all rankers were constrained to the same 10 images on each page. We decided to stay closer to our intended scenario, which likely explains the low Krippendorff's $\alpha$. Nevertheless our algorithm was (mostly strongly) correlated with 80% of the rankers. Going forward we will tune our approach further.

**Web Page Topic Classification**

Our final full-module implementation example for the Arc-Spread architecture is a Web page topic classifier. This module resides in Figure 10’s machine room layer, and utilizes our Hadoop cluster. The input to the classification module is a stream of Web pages, and an integer $n$ that indicates the number of topic clusters the module is to create. The output is a CSV file that contains important words for each topic as well as the pages that belong to each topic.

A classic example for content classification is to cluster article pages on a news site into politics, sports, business, etc. However, when applied to different sets of pages, more interesting topics can arise. For example, much of the sports news during Hurricane Katrina was the effect of the hurricane on Louisiana football teams (LSU and New Orleans Saints). Pages are classified into topics in a two step process. The first is a preprocessing step where a Pig script extracts important text from each page. This text is then clustered using Latent Dirichlet Allocation (LDA) [6].

In the preprocessing step, we take Web pages as input, from which we extract sections, such as the title and headlines. These details are configurable. One could imagine using the summarization snippets above in this context, although we have not yet experimented with this option. Once important text is extracted, we remove stop words and stop phrases.

The text extracts are then clustered using the Mahout LDA implementation [13]. We modified this implementation to provide more complete incremental results in CSV files that can be ingested more quickly and easily by the spreadsheet layer.

**Related Work**

We describe related work that has not already been referred to in the above sections.

There has been a great deal of research on extending the spreadsheet paradigm. Spreadsheets have been extended to include support for image analysis [21], end user programming [20], and, most frequently, data visualization.

For example, [14, 7, 32] extended spreadsheets to support complex objects in cells, such as charts and graphs. The commercial product Tableau provides an intuitive drag and drop interface allowing users to create charts and graphs from existing data sources such as spreadsheets.

Chi et al. [7] describe a spreadsheet approach to displaying and exploring information visualizations with large, abstract, multidimensional data sets that are visually represented in multiple ways. Levoy [21] describes a data visualization system based on spreadsheets. Cells in this spreadsheet contain various graphical objects which enhance visualization. None of the prior work considers use of a spreadsheet metaphor for Web archive analysis.

Work on summarizing Web pages deploy techniques that include utilizing the structure of a page along with the content and the context, employing heuristic techniques, using clickthrough data and learning new models for better summarization. Other methods also include visual summarizations, which use thumbnails, internal images etc.

For example, the algorithm described in [1] relies on the structure of hypertext and the way people describe information in it. It concentrates on tailoring coherent snippets for search results and describes an implemented system, InCommonSense. In contrast to the work presented here, [1] use annotation information contained on pages other than the page being summarized. Similarly, [9] use the context of a Web document, which they obtain from the textual content of all the documents linking to the document. To summarize a target Web document, a context-based summarizer selects which pieces of information in the source documents are relevant to the content of the target.

Berger et al. [5] step away from extracting literal snippets to summarize a Web page. Instead, they begin with a bag of words, and train a model to assemble gist snippets from a human generated training set. The page structure is thus not taken into account, as in the work presented here.

Visual summarization techniques like those described in [16] include using internal images and thumbnails that provide a reliable summarization of Web pages with dominant images, and pages with a simple structure. The images can be complimented using external images to help users understand new pages.

Another approach [28] is to enhance Web page summarization by exploring hidden relationships on the Web using knowledge gained from clickthrough data. In our scenario of Web archive research such data will, presumably, rarely be available.

Heuristic techniques like [11] analyze the content and the structure of a Web page. The algorithm first extracts the content from the page. Then tree generation heuristics are used to extract separators, and identify the content relevant to a user’s query. While related, these techniques apply to a different scenario, where user queries are available as indicators of particular search focus.

Gupta and Kaiser [12] take the dual approach to the widely used ‘extractive’ summarization where text snippets are pulled from a Web page to serve as the page’s surrogate. Instead, [12] strip away uninteresting material, to leave behind what is important. This work is also geared towards accessibility goals.

The idea of constructing special user interface level facilities for word analysis was realized in [22], where word analytics...
and visualizations were combined to support the analysis of literature.

Future Work
ArcSpread’s extensible architecture will enable us to add more spreadsheet functions as the need, and available technologies arise. The cell browsers also provide extension opportunities for different types of cell contents. For example, we plan to integrate the open source Internet Archive Wayback Machine as one of the browsing options for cells containing Web pages.

For image summarization and extraction, we plan to use a part of the data that we collected via the user study to train an SVM classifier’s model weights for the image scores. For the important-text extraction and summarization work, we also plan to use a more generic collection during the $tf*idf$ portion of the scoring than the genre of news Web sites we rely on now.

For the summary browsing user level experience we plan to make the text and image collages explorable by allowing users to control the level of summarization. Similarly, the word browser is ready for more inserts that will help illuminate characteristics of Web content.

The machine room level will see the development of appropriate caching policies, and connections to additional archives. Challenges at that level also include a standardization of progress reporting to the spreadsheet layer.

A number of ArcSpread components need user studies to compare with alternative approaches. Fortunately, the open nature of the architecture will allow us to utilize whichever technologies these studies prove to be optimal in our envisioned scenarios.

Conclusion
We explained our ArcSpread architecture for future social science and historical explorations of Web archives. The architecture centers around a spreadsheet metaphor, which we extended to accommodate concerns associated with Web archive exploration. The three layers—visualization, mixture of browsing and user level computation, and underlying distributed computational engine—invite a number of diverse extensions. We demonstrated ArcSpread’s modular capabilities by three example projects: word browsing, Web page summarization, and automated Web page classification.

The spreadsheet engine is available in open source, and is used in two Biology departments for applications other than Web archive exploration. Some of our ArcSpread machine room components have also been contributed to open source.

Future investigations into culture, politics, entertainment, and science will require excursions into Web archives, rather than national and university library basements. ArcSpread is intended as a step towards preparing for that future.

REFERENCES


